Albert Sickler

ATARI 600XL/800XL

praktische tips, programma's, BASIC

Kluwer Technische Boeken

Atari 600XL/800XL

Albert Sickler

ATARI 600XL/800XL

praktische tips, programma's, BASIC

KLUWER TECHNISCHE BOEKEN B.V. DEVENTER - ANTWERPEN

Illustraties: Wim Niessink
Lay-out: Wim Wijnolts

ISBN 90 201 1763 7
D/1985/0108/185

© 1985 Kluwer Technische Boeken B.V. Deventer

1e druk 1985

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm of op welke andere wijze ook, zonder voorafgaande schriftelijke toestemming van de uitgever.

No part of this book may be reproduced in any form, by print, photoprint, microfilm or any other means without written permission from the publisher.

Ondanks alle aan de samenstelling van de tekst bestede zorg, kan noch de redactie noch de uitgever aansprakelijkheid aanvaarden voor eventuele schade, die zou kunnen voortvloeien uit enige fout, die in deze uitgave zou kunnen voorkomen.

Inhoud

1. Meer algemene informatie over de computer: hardware en software . . . 9
2. Een eerste verkenning . . . 20
3. En nu een BASIC-programma! . . . 27
4. Het eerste programma wordt uitgevoerd . . . 32
5. Herstellen van fouten en meer informatie over het toetsenbord . . . 37
6. Over getallen en strings . . . 42
7. INPUT, READ, DATA en RESTORE, PRINT en REM . . . 48
8. Besturingsinstructies . . . 57
9. De FOR...NEXT-instructie . . . 72
10. De array . . . 79
11. Subroutines: GOSUB en RETURN . . . 88
12. Manipulaties met strings . . . 93
13. Geluid op de Atari-computer . . . 102
14. Over het scherm en over kleuren . . . 109
15. Over PEEK, POKE en grote letters . . . 112
16. Over plaatjes . . . 119
17. Machine-instructies . . . 130

Appendix 1. Symbolen, schrifttekens en ASCII-codes . . . 145
Appendix 2. BASIC-functies . . . 150
Appendix 3. BASIC-commando's en -instructies . . . 153
Appendix 4. MODE's en kleuren . . . 160
Appendix 5. MODE's, geheugen en schermafmetingen . . . 162
Appendix 6. Nuttige geheugenadressen . . . 163
Appendix 7. Foutboodschappen . . . 165
Appendix 8. Machine-instructies . . . 170

Programma's . . . 176

Trefwoordenregister . . . 189

Woord vooraf

Wat is toch de reden dat computers zo'n geweldige aantrekkingskracht op de mens uitoefenen?
Het antwoord is ongetwijfeld dat de mens in de computer een beetje van zichzelf herkent. Zelfs het eenvoudige programma waarbij een kind zijn eigen naam in kan voeren, waarbij die naam dan vervolgens weer wordt uitgevoerd met het woordje HALLO ervoor, sorteert al dat zo merkwaardige effect... dit is geen machine... dit is 'een beetje mens'.
Soms loop ik door een warenhuis en elke keer sta ik weer versteld van de menigte die de afdeling 'computers' bezet. En weer denk ik dan... het is dat 'beetje mens' dat al die lieden zo biologeert.

Eén van de belangrijkste firma's die aan de massale invoering van computers heeft bijgedragen is ongetwijfeld Atari. Zij waren het die als eerste pure spelletjes-computers op de markt brachten, waarbij het spelletje 'Pong' van Nolan Bushell stellig als klassieker in de geschiedenisboekjes zal worden vermeld.
Na de introductie van de Apple en de PET begreep Atari dat naast de spel-computer ook de personal computer een gouden toekomst had. Atari bracht toen twee modellen uit: de Atari 400 en de Atari 800. Beide modellen gingen uit van dezelfde BASIC, een kenmerk dat trouwens ook voor de nieuwe modellen geldt.
Ondanks het grote succes dat beide modellen op de Amerikaanse markt (en trouwens ook op de Engelse en Duitse markt) behaalden, werden de Atari 400 en 800 niet in Nederland uitgebracht.
Met de Atari 600XL en 800XL komt daar nu verandering in en dat is maar gelukkig ook. De beide machines zijn namelijk zeer fraai en de verkoopsuccessen die nu reeds zijn geboekt, vormen wellicht een nog overtuigender bewijs van de waarde van deze machines.
Atari heeft bij het betreden van de markt duidelijk gekozen voor 'computers voor het gehele gezin'. Voor het bedrijfsleven is het nauwelijks interessant om een computer te gebruiken waar men niet minder dan 256 kleuren op het scherm kan toveren... voor vele spelletjes is dit natuurlijk zonder meer geweldig.
Atari heeft op het gebied van spelletjes trouwens ook een naam hoog te houden. Pacman en Donkey Kong zijn twee voorbeelden van absolute wereldtoppers en het moet worden gezegd, de uitvoeringen op de personal computers zijn buitengewoon fraai. Leidt nu uit deze zinnen niet af dat de Atari primair is gericht op spelletjes. Atari biedt een uitgebreid scala

aan software met vele mogelijkheden waaruit eens te meer blijkt hoe veelzijdig deze computer wel is.
Het nu voor u liggende boek is allereerst bedoeld als reisgids... reisgids voor uw Atari-computer.

Goede reis!

1. Meer algemene informatie over de computer: hardware en software

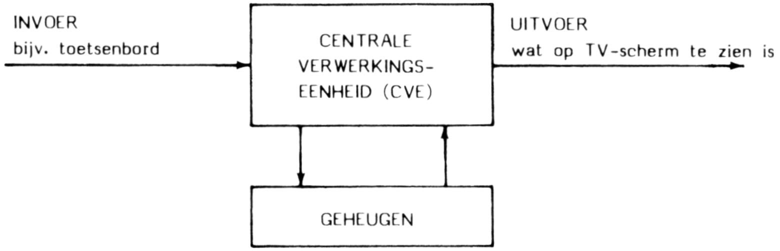
Inleiding

Natuurlijk zouden we nu direct aan de slag kunnen gaan met het bespreken van allerlei programmavoorbeelden. Het lijkt ons echter verstandiger om allereerst eens te bezien wat onze Atari-computer zoal biedt.
Het woord 'computer' betekent in feite 'rekenaar' en eigenlijk is dit een zeer slechte benaming. Uit het woord 'rekenen' zou men namelijk op kunnen maken dat de computer er enkel en alleen maar is om berekeningen uit te voeren en dit is stellig onjuist. Misschien wordt dit idee ons ook wel onwillekeurig opgedrongen; een personal computer lijkt immers een logisch vervolg op de wetenschappelijke pocketcalculator.
Welnu, op pocketcalculators kan men niet met teksten en grafische symbolen werken, zoals dat op computers wel kan. Pocketcalculators kunnen geen kleurrijke spelletjes tonen en ze zijn zeker niet zo algemeen toepasbaar als personal computers.
Inderdaad, allerlei zaken kunnen met onze computer worden gedaan: het bewerken van teksten, het beheren van administraties, het uitvoeren van statistische analyses, het uitvoeren van programma's waarmee leerlingen taal- en rekenonderwijs krijgen, het spelen van allerlei spelletjes en ga zo maar door.
Doorgaans wordt in boeken over computers onderscheid gemaakt tussen de zogenaamde hardware en sotfware van de computer. Wat bedoelt men

nu met termen als hardware en software? Welnu, de hardware is datgene wat bij een computer ook werkelijk tastbaar is. Kortom, alles wat je bij een computer kunt aanraken: kast, toetsenbord, printer enz. behoren tot de hardware van de computer. Een vriend van me omschreef het eens als: 'Alles wat je kapot kunt maken is hardware'.
De software heeft steeds betrekking op de programma's die we met zo'n computer kunnen uitvoeren.
Een programma is in feite een serie instructies waarmee dan een bepaalde bewerking wordt aangegeven, zoals bijvoorbeeld het uitvoeren van één of andere berekening, of het plaatsen van een rij namen in alfabetische volgorde. We zullen zien dat er reeds vele programma's voor de Atari zijn geschreven. Het is dus zeker niet nodig om programmeur te worden om de Atari met plezier te kunnen gebruiken! In de meeste gevallen worden die programma's dan in de vorm van zogenaamde modules verkocht, kleine 'doosjes' die zo in de computer kunnen worden geschoven.
Toch willen we hierbij ook opmerken dat het woord 'programmeren' ons niet zodanig mag afschrikken dat ons zelfs de moed wordt ontnomen het gedeelte over programmeren door te nemen. De praktijk leert immers steeds weer, dat niets zo uitdagend is als de computer naar eigen hand te zetten!

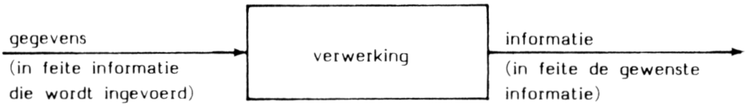
Hardware

In dit gedeelte zullen we dieper op de verschillende hardware-aspecten van onze computer ingaan. We zullen hierbij met name de bijzondere, vaak Engelstalige, termen die we in de folders aantreffen, toelichten. Het gaat er ons hierbij niet alleen om toe te lichten wat bepaalde termen nu wel feitelijk inhouden, maar vooral om aan te geven wat die verschillende mogelijkheden in de praktijk betekenen.
Klaar... daar gaan we!
Het volgende schema toont een algemene voorstelling van onze computer:



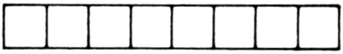
In dit schema zien we allereerst een blok dat de naam 'centrale verwerkingseenheid' draagt. Om nu te kunnen nagaan wat zo'n centrale verwerkingseenheid wel is, bekijken we achtereenvolgens de pijl die naar dit blok toeloopt en de pijl die van dit blok af wijst. Bij de linkerpijl zien we de term INVOER en bij de rechterpijl de term UITVOER. De centrale verwerkingseenheid vormt kennelijk de schakel tussen de INVOER en de UITVOER. Laten we er eens dieper op ingaan.
De pijl waar INVOER bij staat, geeft aan dat er bij onze computer informatie wordt ingevoerd. We kunnen hier bijvoorbeeld denken aan de pro-

gramma's en de gegevens die via het toetsenbord kunnen worden ingevoerd. Het toetsenbord is dan ook een voorbeeld van wat men een 'invoerorgaan' noemt; een onderdeel van de computer dat speciaal is gericht op het invoeren van gegevens. Denk nu niet dat het toetsenbord de enige mogelijkheid is om onze computer van informatie te voorzien. In een later gedeelte zullen we ook andere mogelijkheden, zoals de cassetterecorder en de zogenaamde floppy disk, beschouwen.
We staan nu een moment stil bij de pijl waar UITVOER bij staat. Als we het hebben over informatie-uitvoer, dan hebben we het in eerste instantie over gegevens die door de computer als het resultaat van een bewerking op het scherm van de TV worden weergegeven. Op het scherm kunnen we 24 regels van ieder 40 karakters afbeelden. We kunnen hierbij gewone letters en speciale symbolen afbeelden. Bovendien zijn er uitgebreide kleurmogelijkheden, waarmee zeer aantrekkelijke afbeeldingen kunnen worden samengesteld.
De Atari biedt echter nog veel meer mogelijkheden om informatie uit te voeren. Zo zijn er bijvoorbeeld ook mogelijkheden om een aantal tonen tegelijkertijd te laten klinken. Vooral bij veel spelletjes, maar ook bij allerlei educatieve toepassingen, spelen geluiden een voorname rol. Daarnaast zullen we zien dat we onze 'kale computer' (ook wel 'console' genoemd) aan allerlei apparaten kunnen koppelen; apparaten die dan in feite een uitvoerfunctie hebben. Vooruitlopend op die beschouwing kunnen we bijvoorbeeld denken aan de zogenaamde printer. Zo'n printer is een apparaat om gegevens uit te voeren en wel specifiek om letters of andere symbolen op papier af te drukken.
We komen nu weer terug op de vraag, wat nu wel de centrale verwerkingseenheid (afgekort: CVE, of in het Engels CPU) is. Welnu, de centrale verwerkingseenheid is kennelijk de eenheid, die gegevens verwerkt om zodoende tot de gewenste uitvoer te komen. In verschillende boekjes over informatica wordt dit als volgt aangeduid:



De verwerking van de gegevens vindt plaats op grond van een programma; een reeks instructies die in het geheugen van de computer is opgeslagen. Dit is de reden dat in de eerste afbeelding nog een blok, GEHEUGEN, met pijlen van en naar de CENTRALE VERWERKINGSEENHEID, is geplaatst. Onze computer is een zogenaamde 'digitale computer'. Dit houdt in, dat alle gegevens en alle instructies van een programma door middel van specifieke series enen en nullen worden aangegeven. Ook ingevoerde en uitgevoerde informatie wordt door middel van series enen en nullen voorgesteld. Op het eerste gezicht is het eigenlijk onvoorstelbaar dat we met zo'n enen- en nullen-computer zoveel verschillende zaken kunnen uitvoeren. We kunnen er immers mee rekenen, tekstverwerking mee bedrijven en er allerlei spelletjes op spelen. Misschien denkt u nu, dat voor het werken met zo'n digitale computer dan ook kennis rond die enen en nullen noodzakelijk is. Gelukkig is dat niet zo; het enen- en nullengedrag wordt in feite vrijwel volledig gemaskeerd. We kunnen gewoon teksten

intypen en we zullen straks zien dat een programma ook een tekst is, die we als zodanig kunnen intypen. We moeten daarbij echter wel bedenken dat elke toetsaanslag in zo'n serie enen en nullen wordt omgezet... maar daar merkt u niets van.
Hetzelfde geldt voor de uitgevoerde informatie. Al deze informatie wordt intern door middel van series enen en nullen gerepresenteerd, maar er zijn gelukkig speciale schakelingen waardoor we weer direct herkenbare letters en symbolen op onze TV zien.
Ook de instructies, die we straks zullen intypen, zijn in begrijpelijke taal verwoord. Zo zullen we zien, dat we bijvoorbeeld een term als PRINT gebruiken om aan te geven dat er iets moet worden uitgevoerd (geprint) en INPUT (invoer) als er een gegeven moet worden ingevoerd.
Deze termen worden dan automatisch omgezet in de zogenaamde elementaire enen- en nulleninstructies, waar de centrale verwerkingseenheid zich uiteindelijk van bedient. Deze elementaire instructies noemen we trouwens machinecodes, of ook wel machine-instructies.
De machine-instructies worden uitgevoerd door een centrale verwerkingseenheid die wordt aangeduid met het typenummer 6502. Computerdeskundigen zeggen dan: 'de processor van deze computer is de 6502'.
In feite is zo'n centrale verwerkingseenheid een complexe elektronische schakeling die als één geheel op een klein plakje silicium is gemaakt. Een dergelijke complexe schakeling op een klein plakje silicium noemen we ook wel een chip. Welnu, als zo'n centrale verwerkingseenheid in chipvorm is gemaakt, spreken we van een microprocessor en ten slotte: als een computer gebruik maakt van een microprocessor spreken we van een microcomputer.
Het is interessant om te weten dat onze centrale verwerkingseenheid (CVE) steeds rijtjes van acht enen en nullen tegelijkertijd kan verwerken. Deze eigenschap wordt dan aangeduid met '8-bit processor'. De CVE wisselt voortdurend informatie uit met het geheugen. Zowel de CVE als het geheugen bevat daartoe zogenaamde registers die bestaan uit acht vakjes waarin men een 1 of een nul kan opslaan:



Iedere cel van zo'n register heeft, zoals men dat noemt, een opslagcapaciteit van één bit. Men gebruikt ook wel eens de term byte: één byte komt overeen met acht van deze vakjes, kortom met zo'n register. Het geheugen van onze computer bestaat in feite uit een grote verzameling van dergelijke registers. Het gedeelte van het geheugen waarin we op willekeurige plaatsen gegevens kunnen opslaan, duidt men gewoonlijk aan met RAM-geheugen. De term RAM komt van het Engelse 'Random Access Memory', hetgeen zoveel betekent als: willekeurig toegankelijk geheugen. De Atari 600XL beschikt standaard over een RAM-geheugen van circa 16000 bytes. In de folders wordt dit aangegeven met de uitdrukking 16 Kbyte. Het geheugen van de 600XL kan dan nog tot 64 Kbyte worden uitgebreid. De Atari 800XL beschikt standaard over 64 Kbyte RAM-geheugen en dit kunnen we eventueel tot 128 Kbyte uitbreiden.
Naast het RAM-geheugen beschikt onze computer nog over een ROM-geheugen. Dit geheugen heeft de bijzondere eigenschap dat er alleen maar informatie kan worden uitgelezen. De informatie, die in dit ROM-

geheugen is opgeslagen, is er namelijk door de fabrikant van tevoren onuitwisbaar in opgeslagen. Deze onuitwisbare programma's maken dat we plezierig met onze computer kunnen werken. Ze kunnen immers niet verloren gaan bij een eventuele menselijke fout. De term ROM komt overigens van het Engelse 'Read Only Memory', hetgeen zoveel betekent als: lees-alleen-geheugen. Hoewel het nauwelijks van belang is, vermelden we dat onze Atari standaard met 24 Kbyte ROM-geheugen is uitgerust. Dit ROM-geheugen is tevens uit te breiden en wel met speciale modules die we zo in de computer kunnen schuiven. Deze modules worden in de folder aangeduid als 'Programmamodules'. Wellicht rijst hier de vraag: 'Is het ROM-geheugen dan niet groot genoeg?'. Het antwoord op die vraag luidt: 'Het ROM-geheugen van de computer is uiteraard groot genoeg'. De eventuele uitbreiding heeft echter te maken met de mogelijkheid, de computer op eenvoudige wijze van speciale kant-en-klare programma's te voorzien. We kunnen hierbij als voorbeeld denken aan de vele spelletjes, zoals PACMAN, DONKEY KONG en TENNIS, die in de vorm van dergelijke modules verkrijgbaar zijn.
Het standaard aanwezige ROM-geheugen bevat onder andere een programma, dat de BASIC-instructies die we straks zullen intoetsen, omzet in de machinecodes die uiteindelijk door de CVE kunnen worden uitgevoerd. Dit in ROM opgeslagen 'vertaalprogramma' duiden we aan met de naam 'BASIC-interpreter', of ook wel 'BASIC-vertolker'.
Zo is BASIC de taal die altijd aanwezig is en waar we geen speciale handelingen voor behoeven te verrichten om er mee te kunnen werken. Dat BASIC echter niet de enige taal is waarin we kunnen programmeren, zal duidelijk blijken.

Nog meer over de aansluitmogelijkheden en over de mogelijke uitbreidingen van de Atari 600XL/800XL

In het voorafgaande kwamen al verschillende aansluitmogelijkheden aan de orde. Eén van de kenmerken van de Atari is inderdaad dat er veel mogelijkheden voor zowel in- als uitvoer van gegevens zijn.
In het nu volgende zullen we enkele van die mogelijkheden nog eens nader bezien.

Cassetterecorder

Op een cassetterecorder kunnen we eenvoudig en goedkoop gegevens, bijvoorbeeld een programma, vastleggen. Er kunnen gewone cassettes worden gebruikt. Een eenmaal op cassette weggeschreven programma kunnen we naderhand op commando weer inlezen.
Atari heeft zelf een cassetterecorder uitgebracht: de Atari 1010 programmarecorder. Alleen op deze recorder kunt u zelfgemaakte programma's vastleggen of van voorbespeelde bandjes gebruik maken. Standaard cassetterecorders zijn voor de Atari ongeschikt. Wel kunt u standaard cassettes gebruiken. We raden dan echter wel aan ferro-cassettes en geen chroomdioxide-cassettes te nemen.

Floppy disk unit

Een floppy disk unit is een apparaat dat we met een grammofoon kunnen

De Atari met randapparatuur.

vergelijken. De grammofoonplaten zijn nu de zogenaamde floppy disks (of diskettes). Dit zijn buigzame ('floppy') plastic plaatjes ('disks') die altijd in een papieren hoesje zijn opgeborgen. De floppy disk bevat een magnetisch laagje, waarop we net als bij een bandrecorder informatie kunnen vastleggen. Het gebruik van de floppy disk laat zich ook vergelijken met de cassetterecorder. Het grote verschil met de cassetterecorder is echter de snelheid. Als we informatie op cassette willen opslaan kan dit minuten duren, terwijl het bij de floppy disk een zaak van slechts enkele seconden is! Willen we onze Atari bijvoorbeeld voor professionele administratieve doeleinden gebruiken, dan is het gebruik van de floppy disk min of meer noodzakelijk. Voor de Atari-computers wordt de Atari 1050 floppy disk unit op de markt gebracht. De geheugencapaciteit van de floppies bedraagt 127 Kbyte. Maximaal kunnen er vier van deze floppy units (ook wel 'drives' genoemd) worden aangesloten.

Printer

De printer is een uitvoerapparaat bij uitstek. We merken dit op omdat de cassetterecorder en de floppy disk zowel mogelijkheden bieden om informatie in- als uit te voeren. Deze apparaten kunnen dus zowel als invoer- en als uitvoerapparaat worden beschouwd. Bij de printer geldt dat niet: de printer dient alleen voor de uitvoer. De printer is een apparaat om gegevens op papier af te drukken (te 'printen').
Er zijn een aantal printers door Atari ontwikkeld. We behandelen ze kort:

ATARI 822: een thermische printer. Hierbij wordt ieder teken als een

verzameling puntjes afgebeeld. We spreken in zo'n geval van een (dot-)matrix-printer.

ATARI 1020: dit is een printer/plotter. Het grote verschil tussen een printer/plotter en een printer is dat de eerste ook figuren kan tekenen ('plotten'). De Atari 1020 kan in vier kleuren schrijven (printen) en tekenen (plotten).

ATARI 1027: dit is een daisy-wheel-printer. Het woorde 'daisy' betekent 'margriet'. De reden van deze benaming is dat van een zogenaamd letterwiel gebruik wordt gemaakt waarvan de vorm kan worden vergeleken met de vorm van een margriet. De letters worden hier als geheel op het papier gedrukt, waardoor de kwaliteit aanmerkelijk beter is dan de gebruikelijke kwaliteit die matrix-printers leveren.

Overigens kan men ook niet-Atari-printers gebruiken. In dit geval dient u eerst óf een RS232-interface of een Centronics-interface aan te schaffen. Deze interfaces zijn opgenomen in de Atari Expansion Box.

Uitbreidingspoort
Aan de achterkant van de Atari bevindt zich een brede gleuf ('parallel bus'). Hier bevindt zich een connector met veel aansluitpunten. We spreken hier ook wel van de 'XL Expansion Port'. Het woord 'expansion' betekent uitbreiding. Deze connector biedt inderdaad de mogelijkheid onze Atari op verschillende manieren uit te breiden, bijvoorbeeld om het standaardgeheugen uit te breiden. Zeker als we de 'Expansion Box' gebruiken, een kast waarin tot vijf uitbreidingskaarten kunnen worden gestoken, worden onze mogelijkheden groot.

Monitor
Deze connector biedt u de mogelijkheid om in plaats van een TV een monitor te gebruiken. Monitors zijn beeldschermapparaten die speciaal voor computers zijn ontwikkeld. Ze bieden een scherper beeld, hetgeen uiteindelijk minder vermoeiend is voor de ogen.

Cartridge poort
Boven op de computer bevindt zich een gleuf waarin wij programmamodules kunnen steken (altijd met de netspanning uit). Er zijn tientallen modules (cartridges) door Atari uitgebracht. Bedenk dat ook de cartridges die voor de 400/800-modellen zijn ontwikkeld op de 600XL/800XL kunnen worden gebruikt.

Overigen
Naast de genoemde mogelijkheden biedt Atari nog veel meer. We noemen:

Atari Touch Tablet: een vlak waarop men met een speciale pen tekeningen kan maken. De tekeningen worden dan op het beeldscherm (monitor of TV) weergegeven. Op deze manier kunt u ook grafieken 'inlezen'.

U plaatst de in te voeren grafiek op het Touch Tablet en beweegt vervolgens met de pen over de in te lezen curve.

Atari Track Ball: het Engelse werkwoord 'to track' betekent 'volgen'. De Track Ball is een alternatief voor een joy-stick. Door met de hand de bal te draaien geven we de computer informatie overeenkomstig een joy-stick.

ROM-packs.

Atari Numeriek Toetsenbord: een afzonderlijk toetsenbord voor de invoer van getallen. Zeker als we zeer veel getallen moeten invoeren, bijvoorbeeld bij rekenwerk of administratieve toepassingen, is deze optie interessant.

Atari 850 Interface Module: biedt de mogelijkheid gegevens via één van de vier standaard RS232-poorten dan wel via de Centronics-poort uit te wisselen. Zo kan men verschillende printers aansluiten, of wat misschien nog belangrijker is: een modem.
Een modem is een elektronische schakeling die het ons mogelijk maakt om onze computer via het telefoonnet met andere computers te koppelen. Zo kunnen gegevens van de ene naar de andere computer worden uitgewisseld.

Software

Software is een verzamelnaam voor alles wat met programma's te maken heeft. Als we het hebben over programma's die in ROM zijn opgeslagen, als we het hebben over BASIC of PASCAL, als we het hebben over machine-

instructies, dan hebben we het over software. De werkelijke kracht van een computer wordt voor het grootste gedeelte juist door de software bepaald. Al is de hardware nog zo geavanceerd, zonder software is de computer in feite een nutteloos apparaat.
Beschouwen we de software, dan kunnen we ruwweg drie categorieën herkennen:

a. Alle programma's die te maken hebben met het functioneren van de computer zelf. Alle programma's die in het ROM-geheugen zijn opgeslagen, vallen onder deze categorie.
b. Programma's die door de gebruiker zelf kunnen worden opgesteld. Men duidt deze programma's wel eens aan met de term 'user programs'.
c. Programma's die men kant-en-klaar kan kopen en waarmee men onmiddellijk kan werken.

We zullen deze categorieën in het nu volgende afzonderlijk behandelen.

Ad a. Programma's voor het functioneren van de computer zelf
Bij een eerste kennismaking komt het waarschijnlijk tamelijk vreemd over, dat de computer voor het functioneren zelf allerlei programma's nodig heeft. Toch is dat zo. Vrijwel alle delen, zoals toetsenbord, cassetterecorder, printer enz. worden door afzonderlijke programma's bestuurd. In het voorafgaande merkten we reeds op dat onze computer is uitgerust met 24 Kbyte ROM-geheugen. Een groot gedeelte van dit ROM-geheugen wordt ingenomen door de zojuist genoemde programma's. Een ander gedeelte wordt ingenomen door een programma dat onze BASIC-instructies in voor de computer verteerbare machine-instructies omzet. Dit gedeelte duiden we aan met 'BASIC-interpreter'. Deze besturingsprogramma's vormen met nog andere programma's een pakket dat we aanduiden met de naam Operating System (beheerssysteem).

Ad b. Programma's die door de gebruiker zelf kunnen worden opgesteld
Een groot gedeelte van deze handleiding geeft een inleiding tot het programmeren in BASIC. BASIC is een hogere programmeertaal, hetgeen inhoudt dat we met behulp van BASIC allerlei programma's kunnen opstellen, die direct gericht zijn op de door de computer te klaren klus. De term 'hogere' in de aanduiding 'hogere programmeertaal' schrikt wellicht velen af, omdat onwillekeurig een verband wordt gelegd met 'hogere' wiskunde. Deze vergelijking gaat gelukkig in het geheel niet op. In dit geval wil de term 'hogere' eerder zeggen dat de manier waarop we de computer instructies geven, eerder overeenkomt met een gewone taal, zodat het programmeren dan doorgaans eenvoudig verloopt.
Bij het programmeren in BASIC worden we inderdaad op geen enkele wijze gehinderd door de specifieke kenmerken van onze digitale computer. Zo merken we bijvoorbeeld tijdens het programmeren niet dat het geheugen is opgebouwd uit registers waarin alleen maar rijtjes enen en nullen worden opgeslagen. Standaard beschikt de Atari over BASIC. Het eerste gedeelte van dit boek is volledig gericht op het werken met BASIC. Er komen evenwel een aantal interessante uitbreidingsmogelijkheden. De belangrijkste mogelijkheden worden in het nu volgende opgesomd.

LOGO: dit is een uitbreidingsmogelijkheid die onze computer tot een

Verschillende schermbeelden van programma's.

uitermate interessante machine voor kinderen maakt. De opzet van dit programma is ontwikkeld door Seymour Papert en de staf van het Artificial Intelligence Laboratory van het Mekka der techniek... het Massachusetts Institute of Technology (MIT). Ook in Nederland bestaat bijzonder veel belangstelling voor LOGO.

PASCAL: PASCAL is evenals BASIC een hogere programmeertaal. Vooral in het voortgezet onderwijs wordt meer en meer aandacht besteed aan deze programmeertaal en wel omdat er een aantal krachtige instructies wordt geboden, die het mogelijk maken om tot zeer overzichtelijke programma's te komen.

ASSEMBLER: een programmapakket waarmee het mogelijk is direct in machine-instructies te programmeren. In het voorafgaande stelden we reeds dat de computer uiteindelijk alleen maar machine-instructies verwerkt. Ook de besturing van de verschillende delen van de computer geschiedt met behulp van deze machine-instructies. Dit houdt in dat we met behulp van dit programmapakket directe toegang kunnen krijgen tot alle delen van de computer. Een goede assembler-programmeur voelt zich werkelijk heer en meester over zijn computer!
We willen de term 'assembler' nog nader toelichten. Machine-instructies zijn in feite alleen maar rijtjes enen en nullen. Het zal duidelijk zijn dat het programmeren in alleen maar rijtjes enen en nullen een tamelijk hoofdpijnverwekkende aangelegenheid is. Vandaar dat men korte, eenvoudig te onthouden lettercodes voor deze instructies heeft ontwikkeld. Het programmeren in deze korte codes heet assembler-programmeren en het programma dat deze lettercodes in machine-instructies omzet, heet ASSEMBLER. Voor de beginnende programmeur is dit zonder twijfel moeilijke materie.

PILOT: een taal die speciaal is ontworpen om programma's voor computer ondersteund onderwijs (COO) op te stellen. Er zijn speciale mogelijkheden in PILOT om allerlei fraaie plaatjes samen te stellen.

Ad c. Kant-en-klaar programma's

Nogmaals, u hoeft zelf geen programmeur te worden om van de computer gebruik te kunnen maken. Voor veel toepassingen zijn al programma's geschreven of worden op dit moment programma's ontwikkeld. Deze programma's worden dan op één van de volgende wijzen geleverd:

- Afgedrukte tekst: we spreken in dit geval van een listing. Dit boek bevat veel van dit soort programma's.
- Cartridges: zoals reeds genoemd gaat het hier om ROM-modules die al kant-en-klare programma's bevatten. De meeste cartridges hebben met spelletjes te maken.
- Voorbespeelde cassettes: deze zijn doorgaans goedkoper dan cartridges. Veel kleine bedrijfjes leggen zich toe op het maken van spelcassettes.
- Voorbespeelde diskettes: hierbij gaat het meestal om professionele software, bijvoorbeeld een boekhoudpakket of een pakket voor de salarisadministratie.

2. Een eerste verkenning

Aanzetten van de computer

In dit hoofdstuk beschrijven we een eerste kennismaking. Als we met kloppend hart de computer uit de verpakking nemen zien we naast de computer nog een aantal losse onderdelen.
Allereerst is er het netsnoer met het afzonderlijke 'kastje' dat een transformator bevat. Het ene uiteinde van het snoer steken we in het stopcontact en het andere uiteinde (met de ronde stekker) steken we in de zijkant van de computer waar PWR. IN bij staat. Let op: in deze ronde stekker zit een gleuf waaruit blijkt hoe deze stekker precies in de computer moet worden gestoken.
Dan is er nog een tweede losse kabel. Eén uiteinde daarvan past in de antenne-ingang van onze TV en het andere past op de uitgang van de computer waar TV bij staat. Het gaat om de uitgang waar een klein rond metalen busje naar buiten steekt. Hierop past alleen maar de stekker van onze kabel; vergissen is dus niet mogelijk.
De TV hebben we op kanaal VHF 4 afgestemd. We zetten nu de aan/uitschakelaar aan die zich links aan de achterkant van de computer bevindt (hierbij staat de term PWR. ON).
Als de computer aanstaat zal het rode lampje bij het toetsenbord gaan branden.
Regel nu de TV af, zodanig dat een optimaal beeld wordt verkregen. Op het scherm zien we:

```
READY
■
```

De term READY geeft aan dat onze computer klaar staat om bevelen te

ontvangen. Het vakje is de zogenaamde cursor. We zullen zien hoe we de cursor onder andere gebruiken om een bepaalde plaats op het beeldscherm aan te wijzen.
Alles klaar...? Daar gaan we!

Voorbeeld 1

We toetsen in:

```
PRINT 45+37
```

Merk op dat het vierkantje zich steeds verplaatst. Het geeft kennelijk de plaats aan waar de tekst tijdens het intypen verschijnt. De spatie tussen PRINT en 45 verkrijgen we door op de brede spatiebalk te drukken. Vervolgens drukken we op de toets waar RETURN op staat. Onmiddellijk hierna verschijnt:

```
82
```

en wederom:

```
READY
```

en ten slotte weer de cursor. (N.B. Als u een fout heeft gemaakt, of er verschijnt een vreemde tekst, toets dan de uit te voeren opdracht opnieuw in.)
We zien dus dat de computer de aangegeven optelling, na de term PRINT, direct heeft uitgevoerd. Op deze wijze kunnen we de computer kennelijk op dezelfde eenvoudige wijze gebruiken als een pocketcalculator.
Bedenk hierbij dat u niet zonder meer 45+37 of 45+37= mag intoetsen. U dient altijd met de term PRINT te beginnen; PRINT geeft namelijk aan dat de waarde van de uitdrukking rechts van de term, met andere woorden 45+37, moet worden afgedrukt. PRINT is een van de standaardtermen uit de programmeertaal BASIC.
De boodschap 'READY' geeft aan dat u weer een nieuwe som kunt intoetsen.

Voorbeeld 2

Stel u heeft 575 dollar en u wilt weten met hoeveel gulden dit overeenkomt als de koers 2,63 is. We toetsen in:

PRINT 575*2.63 *(gevolgd door de RETURN-toets)*

en onmiddellijk hierna toont de computer:

```
1512.25
```

Bij dit voorbeeld moeten we enkele belangrijke opmerkingen plaatsen. De eerste opmerking betreft het feit dat een getal als 2,63 nu als 2.63

wordt ingetypt. Met andere woorden, de komma in het getal is vervangen door een punt. Dit gebruik berust op de Engelse schrijfwijze, waar men steeds een punt in plaats van een komma gebruikt. Men spreekt dan van een 'decimal point'. Onthoud dus:

In plaats van de komma in een getal gebruiken we altijd de punt.

De tweede opmerking betreft het feit dat bij onze vermenigvuldiging in plaats van het zo vertrouwde vermenigvuldigteken x nu het teken * is gebruikt. De reden hiervan is eenvoudigweg dat we op deze manier iedere mogelijke verwarring tussen de letter X en het vermenigvuldigteken x vermijden. Onthoud dus:

> Om iedere mogelijke verwarring met de letter X en het vermenigvuldigteken x te voorkomen, gebruiken we steeds het teken
>
> *
>
> als vermenigvuldigteken.

Om dezelfde reden zien we op het beeldscherm een streep door het cijfer 0. Zo wordt iedere mogelijke verwarring met de letter O voorkomen.

Voorbeeld 3

In een warenhuis worden op vier achtereenvolgende dagen 18, 24, 17 en 29 personal computers verkocht. Hoeveel personal computers worden er gemiddeld per dag verkocht?
Het gemiddelde wordt, zoals bekend, gevonden door de verschillende getallen op te tellen en vervolgens de som van deze getallen door het aantal getallen te delen. We toetsen in:

```
PRINT (18+24+17+29)/4
```

Voor alle duidelijkheid: het teken (staat boven het cijfer 9, we dienen dit teken dus met ingedrukte SHIFT-toets in te typen. Het teken + staat onder op de toets en wordt zonder SHIFT-toets ingetoetst.
Onmiddellijk na het indrukken van de RETURN-toets verschijnt:

```
22

READY
```

Laten we nu eens nagaan hoe we in BASIC het gemiddelde hebben uitgerekend. Allereerst moesten we de som van de getallen bepalen en dit wordt aangegeven met:

```
(18+24+17+29)
```

Merk op dat we haakjes gebruiken en wel op de manier zoals we gewend zijn. De deling door 4 wordt aangegeven door achter bovengenoemde

uitdrukking een schuine deelstreep en een 4 te plaatsen. Onthoud:

Een deling wordt altijd door de schuine deelstreep / aangegeven.

De achtergrond van de notatie van breuken met een schuine deelstreep is, dat we op deze wijze een breuk altijd als een uitdrukking op één regel kunnen weergeven.
In de volgende tabel zien we nog een aantal rekenkundige uitdrukkingen, waarbij steeds de overeenkomstige BASIC-uitdrukking wordt gegeven.

uitdrukking	**uitdrukking in BASIC**
$\frac{2}{5+4}$	2/(5+4) N.B. We moeten nu wel haakjes gebruiken, anders stond er 2/5+4 en dat komt overeen met $\frac{2}{5}+4$.
$\frac{41+37}{65+48}$	(41+37)/(65+48) Haakjes vormen wederom de oplossing.
$\frac{6(43+17)}{18+57}$	6*(43+17)/(18+57) N.B. Niet vergeten het vermenigvuldigteken tussen 6 en (43+17) te plaatsen!
$\frac{25}{127+\frac{31}{16+53}}$	25/(127+31/(16+53)) Wederom vormen haakjes de oplossing.

Voorbeeld 4

Een stuk grond met een oppervlakte van 500 m^2 heeft precies de afmetingen van een vierkant. Hoe lang is een zijde? Voor een vierkant geldt de bekende formule:

zijde x zijde = oppervlakte

of

$\text{zijde}^2 = \text{oppervlakte}$

dus:

$\text{zijde} = \sqrt{\text{oppervlakte}}$

We kunnen dus uit de oppervlakte de zijde berekenen door de wortel uit de oppervlakte te trekken. Bij de meeste pocketcalculators verloopt die procedure als volgt: a. toets een getal in; b. druk op de toets met

het wortelteken. Het wortelteken, en meer in het algemeen, iedere aanduiding van een functie, wordt in BASIC met een korte term aangegeven. Voor de wortelfunctie wordt de term SQR gebruikt. Deze term is afkomstig van het Engelse 'SQuare Root' (vierkantswortel). Na de term SQR plaatsen we dan, tussen haakjes, het getal waarop de functie betrekking heeft. We toetsen in:

```
PRINT SQR(500)
```

Resultaat na het indrukken van de RETURN-toets:

```
22.3606797
```

SQR is slechts één van de vele mogelijke functies die we in BASIC direct kunnen aangeven. De volgende tabel geeft een overzicht van de verschillende mogelijkheden.

BASIC-aanduiding	**algebraïsche notatie**	**betekenis**
ABS(X)	(X)	geeft de absolute waarde van X. Dit is de waarde zonder puls- of minteken.
ATN(X)	arctan(X)	geeft de arctangens van X. Uitkomst tussen -pi/2 en pi/2.
COS(X)	cos(X)	geeft de cosinus van X; X in radialen.
EXP(X)	e^X	geeft de e-macht van X.
INT(X)	geen	rondt de waarde van X altijd naar beneden af. Zo geeft INT(3.9) de waarde 3 en INT(-3.1) de waarde 4. We kunnen een getal 'echt' afronden met behulp van INT(X+0.5).
LOG(X)	^{e}ln(X)	geeft de zogenaamde natuurlijke logaritme van X. De 10-log verkrijgen we door deling met LOG(10).
SGN(X)	geen	geeft de waarde -1, 0 of 1 en wel afhankelijk van het feit of X negatief, 0 of positief is.
SIN(X)	sin(X)	geeft de sinus van X; X in radialen.
SQR(X)	$\sqrt{X}$	geeft de wortel uit X.

Een volledig overzicht treffen we aan in appendix 2. De meeste functies zijn met name voor wetenschappelijke berekeningen van belang en menigeen zal een dergelijke functie nooit gebruiken. Voor deze laatste categorie is bovenstaande tabel dan ook weinig interessant.
Merk op dat de notatiewijze voor alle functies gelijk is: eerst geven we met de term de functie aan en hierna plaatsen we tussen haakjes het getal waarop deze functie betrekking heeft. Dus:

TERM(getal)

Het getal tussen haakjes noemen we ook wel eens het argument van de functie en de waarde die wordt berekend wordt de functiewaarde

genoemd. Binnen de haakjes na zo'n functieterm mogen we ook nog uitdrukkingen schrijven. Zo mogen we bijvoorbeeld ook intoetsen:

```
PRINT SQR(125/(2+3))
```

We mogen zelfs meer functies aangeven, bijvoorbeeld:

```
PRINT SQR(125/(2+SQR(9)))
```

Voorbeeld 5

Een aardig vraagstuk is om eens na te gaan hoeveel verschillende patroontjes van enen en nullen in zo'n register van acht vakjes kunnen worden opgeslagen.

1	0	1	1	1	0	0	1

Eén van de patroontjes die precies in een register van acht vakjes past. De vraag is nu, hoeveel verschillende patroontjes kan men zo opbergen?

Welnu, als men slechts over een register van één vakje beschikt,kan men slechts een 0 of een 1 opbergen, met andere woorden twee patroontjes. Beschikt men over twee vakjes, dan wordt het aantal mogelijkheden twee keer zo groot, met andere woorden het aantal mogelijkheden wordt 2 x 2 = 4. Bij drie vakjes hebben we 2 x 2 x 2 = 8 mogelijkheden. Anders gezegd:

bij 1 vakje	→	2 mogelijkheden
bij 2 vakjes	→	2 x 2 mogelijkheden
bij 3 vakjes	→	2 x 2 x 2 mogelijkheden
bij 4 vakjes	→	2 x 2 x 2 x 2 mogelijkheden
...	→	...
bij n vakjes	→	$\underbrace{2 \times 2 \times 2 \ldots 2}_{\text{n maal}}$ mogelijkheden

Een uitdrukking als 2 x 2 x 2 ... 2 (met precies n tweeën) schrijft men ook wel als:

$$2^n$$

en men spreekt dan van de n-de macht van twee. Een uitdrukking als 2^n noteren we op de computer steeds als volgt:

```
2^n
```

Om ons vraagstuk rond het aantal codes, dat we in een register met acht vakjes kunnen plaatsen, te kunnen oplossen toetsen we uiteindelijk in:

```
PRINT 2^8
```

N.B. Het teken ^ staat rechts boven de toets met het teken *. Om dit

teken (^) in te toetsen drukken we wederom eerst de SHIFT-toets in en we houden deze toets ingedrukt tijdens het intoetsen van *. Na het indrukken van de RETURN-toets zien we:

```
256
```

Het gaat hier met recht om een zeer groot aantal codes. We zullen straks zien dat de computer in feite maar een klein aantal codes gebruikt.

Voorbeeld 6

In onze tot nu toe besproken voorbeelden ging het om berekeningen en zo wordt wederom het idee gevestigd dat we met een computer alleen maar kunnen rekenen. Het volgende zeer eenvoudige voorbeeld toont dat we in ieder geval ook teksten kunnen afdrukken. In een latere fase zullen we zien dat we ook allerlei manipulaties met tekst kunnen uitvoeren. We toetsen in:

```
PRINT "EEN GOEDE MORGEN"
```

Het aanhalingsteken bevindt zich boven de 2-toets. In dit geval moeten we dus weer de SHIFT-toets gebruiken. Onmiddellijk na het indrukken van de RETURN-toets verschijnt:

```
EEN GOEDE MORGEN
```

We zien dus dat de tekst, zoals die in de PRINT-instructie verschijnt, zonder aanhalingstekens wordt afgedrukt.

3. En nu een BASIC-programma!

Als we aan een computer denken, denken we allereerst aan programmeren. Een programma is een serie instructies om de computer aan te geven wat voor bewerkingen moeten worden uitgevoerd om een bepaald resultaat te verkrijgen. Als u zelf zo'n reeks instructies opstelt bent u, zoals dat heet, 'aan het programmeren'.
Programmeren schrikt velen af en in feite is dat zeer ten onrechte. Als u deze eerste hoofdstukken zorgvuldig doorleest, zult u stellig ontdekken hoe eenvoudig alles in zijn werk gaat.
In dit gedeelte zullen we een realistische toepassing, en wel een renteberekening, behandelen. Het gaat om het berekenen van een toekomstige waarde als we een zeker kapitaal gedurende een jaar tegen een zekere rente laten uitstaan. Uiteraard gaat het hier om een eenvoudig voorbeeld, want het gaat ons nu om de principes van het programmeren.
Allereerst zullen we de te gebruiken formule uitleggen. Maak hier nu niet uit op, dat gebruikers van computers zelf formules moeten opstellen. In 99% van de gevallen is zo'n formule al voorradig. Voor onze renteberekening geldt dit in feite ook; in ieder boekje over financiële berekeningen treffen we deze formule aan.
Alvorens we beginnen geven we nog een kleine waarschuwing. Lees dit hoofdstuk eerst zorgvuldig door en toets nog niets in. Het volgende hoofdstuk (4) toont hoe we het in dit hoofdstuk te behandelen programma ook werkelijk kunnen intoetsen en... gebruiken, dus:

Typ nu nog niets in. Het intoetsen en werken met het programma wordt in het volgende hoofdstuk besproken.

Klaar... daar gaan we.
Als we een kapitaal K tegen een rente van R procent uitzetten, zal de waarde W na afloop van een jaar overeenkomen met:

$$W = K \times (1 + \frac{R}{100})$$

Bijvoorbeeld, als u *f*.1000,-, met andere woorden K = 1000, tegen 8%, met andere woorden R = 8, precies 1 jaar laat uitstaan, wordt de waarde W:

$$W = 1000 \times (1 + \frac{8}{100}) =$$

$$= 1000 \times (1 + 0.08) = 1080$$

Onze formule zal wel geen problemen opleveren. Nogmaals, het gaat ons nu om een introductie in BASIC en niet om een introductie in renteberekeningen.
Uitgangspunt, voor het op te stellen programma, is dus de formule:

$$W = K \times (1 + \frac{R}{100})$$

Voor het programma zetten we deze formule in de volgende regel om:

```
LET W=K*(1+R/100)
```

De uitdrukking rechts van het teken = zal na het vorige hoofdstuk wel geen problemen meer opleveren. We gebruiken weer het teken * als vermenigvuldigteken en de schuine deelstreep / om de uitdrukking op één regel te noteren.
Nieuw is het woord LET. LET is een standaardwoord in BASIC, dat zoveel betekent als 'laat ... gelijk zijn aan ...' of in ons voorbeeld:

Laat W gelijk zijn aan de uitdrukking rechts van het teken =

We zullen straks zien dat we de zin die met LET begint gewoonweg kunnen intypen. Een dergelijke zin vormt dan een instructie. Ons programma zal uit meer instructies bestaan en al die instructies te zamen vormen uiteindelijk het programma.
Als de computer de instructie LET W=K*(1+R/100) tegenkomt en de aanduiding W is nog niet eerder gebruikt, dan zal het volgende gebeuren:

- Allereerst wordt een gedeelte van het geheugen gereserveerd;
- dit geheugengedeelte wordt vervolgens 'gelabeld' (bestempeld) met de naam W;
- de waarde die uit de uitdrukking rechts van het teken = volgt, wordt berekend en in dit gelabelde geheugengedeelte opgeslagen.

Op dit moment kunnen we nog niet nagaan wat die waarde zal zijn, de grootheden K en R hebben immers nog geen waarde gekregen in ons 'programma'. In feite kunnen R en K allerlei waarden aannemen en dat houdt ook in, dat W in principe iedere waarde kan krijgen.
We kunnen dus *verschillende* getallen in zo'n gelabeld geheugengedeelte opslaan en om dat karakter te onderstrepen, spreken we van een *variabele*.
Een variabele is dus niets anders dan een gelabeld gedeelte van het geheugen, dat kennelijk verschillende waarden kan bevatten. Elke variabele in BASIC heeft een naam. Zo wordt in ons voorbeeld de naam W gehanteerd, om de variabele aan te geven die de grootheid 'waarde' representeert. Voor het geven van namen gelden de volgende regels:

- Een naam mag uit een willekeurig groot aantal schrifttekens (letters en cijfers) bestaan.

- Het eerste schriftteken van een variabelenaam dient altijd een letter te zijn.

- Een naam mag geen leestekens bevatten.

Voorbeelden van correcte namen zijn:

W K R WAARDE KAPITAAL RENTE

Voorbeelden van incorrecte namen zijn:

3B *(begint met een cijfer)*
X? *(we mogen geen leestekens gebruiken)*

Bij ons voorbeeld gebruiken we korte namen als W, K en R. In de praktijk kan het de moeite lonen om wat langere namen te kiezen, zodat uit de namen al direct duidelijk wordt, waarop variabelen betrekking hebben. Het maakt een programma in ieder geval overzichtelijker.
Het is duidelijk dat we met onze ene instructie nog geen programma hebben. Alvorens W werkelijk kan worden berekend, dienen K en R van een waarde te zijn voorzien.
Het toekennen van waarden aan variabelen gaat in BASIC zeer eenvoudig. We gebruiken weer LET. We breiden ons programma uit met twee regels (straks zien we hoe we de verschillende regels onder elkaar schrijven).

```
LET K=1000
LET R=8
```

De eerste regel kent dus de waarde 1000 toe aan de variabele K en de tweede regel kent 8 toe aan de variabele R. Gezien het karakter van deze regels spreken we hier van toekenningsinstructies. Het zal duidelijk zijn, dat we er straks voor moeten zorgen dat de computer eerst deze regels moet uitvoeren alvorens de regel die begint met LET W= enz. dient te worden uitgevoerd.
Om nu aan te geven in welke volgorde de verschillende regels moeten worden uitgevoerd, nummeren we iedere regel. Onthoud:

We nummeren iedere regel om aan te geven in welke volgorde de verschillende regels moeten worden uitgevoerd.

Het nummer voor een regel noemen we een regelnummer. Een regelnummer dient steeds een positief geheel getal te zijn. De regelnummers dienen tevens oplopend in grootte te zijn. Uitdrukkelijk stellen we dat de nummers elkaar niet precies hoeven op te volgen, zoals dat bijvoorbeeld wel het geval is bij de reeks 1, 2, 3, 4 enz. Dit is overigens wel een correcte reeks regelnummers, maar deze reeks biedt geen enkele mogelijkheid om naderhand nog instructies toe te voegen (dit zal nog worden besproken). Nummeren we daarentegen met de nummers 10, 20, 30, 40 enz. dan kan men eventueel tussen twee regels nog negen andere regels invoegen. Laten we voorlopig maar afspreken dat we, om deze reden, in onze voorbeeldprogramma's de regels steeds volgens de reeks 10, 20, 30, 40 enz. zullen nummeren. De nummering van de regels zal om twee redenen van groot nut blijken te zijn:

- bij het opmaken cq wijzigen van een programma kan men handig van deze nummers gebruik maken;
- het zal straks blijken dat we in programma's spronginstructies kunnen opgeven. Bij deze spronginstructies dienen de regelnummers als het ware als labels (verwijsadressen) waar naar toe moet worden gesprongen.

We nummeren onze drie instructies als volgt:

```
10 LET K=1000
20 LET R=8
30 LET W=K*(1+R/100)
```

Op deze wijze kan W in ieder geval worden berekend; aan K en R worden nu immers van tevoren waarden toegekend. We hadden natuurlijk evengoed de volgende volgorde kunnen aanhouden:

```
10 LET R=8
20 LET K=1000
30 LET W=K*(1+R/100)
```

De eerste twee instructies kunnen dus worden omgewisseld. Ons programma is nog niet volledig, we moeten namelijk nog een instructie opnemen die tot resultaat heeft, dat de gevraagde waarde W op het scherm wordt afgebeeld. We voegen daarom toe:

```
40 PRINT W
```

De term PRINT zijn we in het voorafgaande reeds tegengekomen. De aanduiding na PRINT geeft aan wat moet worden getoond; in dit geval de waarde van W. De laatste regel van ons programma geeft aan dat het programma af is:

```
50 END
```

Dit is gewoon een weetje, kortom er valt verder niets aan te begrijpen. We vermelden nog dat deze END-instructie eventueel mag worden weggelaten, we zullen dat echter consequent niet doen. Het volledige programma luidt nu:

```
10 LET K=1000
20 LET R=8
30 LET W=K*(1+R/100)
40 PRINT W
50 END
```

Als we dit programma intoetsen (dit wordt in het volgende hoofdstuk getoond) en we geven het zogenaamde RUN-commando, dan zal onze computer de instructies één voor één, in volgorde van de regelnummers afwerken. Het zal nu duidelijk zijn dat een programma in feite niets meer is dan een serie instructies die na een zeker startcommando door de computer automatisch wordt afgewerkt.
Als zodanig verschilt de werkwijze dan ook volledig met de in het vorige hoofdstuk geschetste werkwijze. Daar werden instructies zonder regelnummer ingetypt, zoals PRINT 45+37 en na het indrukken van de RETURN-toets toonde de computer dan onmiddellijk het antwoord.
Bij een programma toetsen we ook instructies in, maar dan wel voorafgegaan door regelnummers. De instructies worden nu opgeslagen in het geheugen zonder dat ze direct worden uitgevoerd. Pas na het intoetsen

van een speciaal commando zal de computer de instructies één voor één uitvoeren.
Uit het gegeven voorbeeld kunnen we al enige wetenschap putten omtrent de vraag hoe we programma's opstellen. In het voorbeeld was een formule het uitgangspunt, namelijk:

$$W = K \times (1 + \frac{R}{100})$$

Deze formule werd in feite gekopieerd (regel 30) en van tevoren werd er wel voor gezorgd dat aan K en R door middel van LET-instructies waarden werden toegekend.
Ten slotte zal men in ieder programma één of meer PRINT-instructies aantreffen. Zonder PRINT-instructies immers geen afbeeldingen van resultaten op het scherm.
Aan het slot van deze eerste kennismaking met het programmeren kunnen we constateren dat de instructies inderdaad zeer eenvoudig zijn. We gebruiken gewone termen zoals PRINT en LET. Op geen enkele wijze worden we eraan herinnerd dat de computer in feite alleen maar met reeksen enen en nullen werkt.
Gelukkig maar!

4. Het eerste programma wordt uitgevoerd

Het uitvoeren van het programma

In dit gedeelte bespreken we het intoetsen en het werken met het programma dat in het vorige hoofdstuk werd geïntroduceerd. We gaan er van uit dat de computer aan staat en het scherm toont als laatste regel:

```
READY
■
```

Nu toetsen we het volledige programma in en iedere regel wordt afgesloten

door op de RETURN-toets te drukken. We toetsen in:

```
10 LET K=1000              (gevolgd door de RETURN-toets)
20 LET R=8                 (gevolgd door de RETURN-toets)
30 LET W=K*(1+R/100)       (gevolgd door de RETURN-toets)
40 PRINT W                 (gevolgd door de RETURN-toets)
50 END                     (gevolgd door de RETURN-toets)
```

Vergeet niet op de SHIFT-toets te drukken als een symbool op een toets boven een ander symbool staat. Als u nu een vergissing heeft gemaakt, bijvoorbeeld in plaats van:

```
20 LET R=8
```

heeft u ingetoetst:

```
20 LRT R=8
```

dan volgt een zogenaamde foutmelding (ERROR...) en u typt de volledige regel gewoon opnieuw in.
De spanning stijgt... We zouden op dit moment het programma werkelijk kunnen laten uitvoeren. Opdat onze computer het programma nu daadwerkelijk gaat uitvoeren, moeten we echter eerst een zogenaamd commando intoetsen. Een commando verschilt op een aantal punten van een instructie.
Ten eerste: we typen nu geen regelnummer in, het commando vormt immers geen instructie in het programma.
Ten tweede: Een commando dient altijd direct door de computer te worden uitgevoerd.

De bekendste commando's zijn:

NEW Als het commando NEW wordt gegeven, wordt een eventueel in het geheugen opgeslagen programma geheel en al gewist. Op dit moment is het onverstandig om dit commando uit te voeren omdat ons zojuist ingetoetste programma dan immers verloren zou gaan.

LIST Toont het programma op het scherm. We noemen de weergave van het programma ook wel eens 'listing'.

RUN Dit is het commando dat de computer opdracht geeft het programma uit te voeren. Men spreekt ook wel eens van het 'runnen' of 'executeren' van een programma. Eventueel kunnen we na RUN nog een regelnummer opgeven om op deze wijze aan te geven, dat het programma vanaf de zo aangegeven regel moet worden doorlopen.

Bedenk dat we, na het intoetsen van een commando, steeds weer op de RETURN-toets moeten drukken. We zullen nu het commando RUN intoetsen, zoals gezegd om de computer aan te geven dat het programma daadwerkelijk uitgevoerd moet worden. We toetsen in:

RUN *(gevolgd door de RETURN-toets)*

Als resultaat verschijnt hetgeen op de volgende pagina te zien is.

```
1080

READY
```

Het getal 1080 komt natuurlijk met de berekende waarde van W overeen. Tevens zien we weer de boodschap 'READY', ten teken dat de computer gereed is om een nieuw commando uit te voeren.
We kunnen bijvoorbeeld nogmaals het commando RUN intoetsen (doen!). Opnieuw zal de getoonde tekst verschijnen. Kortom, zolang het programma in het geheugen van de computer is opgeslagen, kunnen we het 'runnen'. We kunnen uiteraard ook het LIST-commando geven:

LIST *(gevolgd door de RETURN-toets)*

Als resultaat verschijnt op het scherm:

```
10 LET K=1000
20 LET R=8
30 LET W=K*(1 +R/100)
40 PRINT W
50 END
```

We spreken bij een programma dat weer wordt afgedrukt doorgaans van een listing. Na het woord LIST kan men ook nog op verschillende wijzen regelnummers opgeven. Zo kan men aangeven welk gedeelte van het programma moet worden getoond. Voorbeelden:

LIST 20,40	Het programma zal van regel 20 t/m 40 worden getoond.
LIST 30	Alleen regel 30 wordt getoond.

Ten slotte het commando NEW. We meldden reeds dat dit commando de computer de opdracht geeft een eventueel in het geheugen opgeslagen programma te wissen. Wissen is een vrij ingrijpende aangelegenheid, waarmee we maar willen zeggen dat u met het NEW-commando extra voorzichtig moet zijn. We toetsen in:

```
NEW
```

Om na te gaan of het voorbeeldprogramma nu inderdaad is gewist, geven we nogmaals het LIST-commando. De computer toont slechts:

```
READY
```

Met andere woorden: alles is gewist!

Nog wat kleine opmerkingen

■ LET mag worden weggelaten
De term LET mag worden weggelaten: deze term is zoals dat ook wel eens wordt genoemd 'optioneel'. Bijvoorbeeld zoals in het behandelde

programma:

```
10 K=1000
20 R=8
30 W=K*(1+R/100)
40 PRINT W
50 END
```

Programmeurs met enige ervaring zullen gewoonlijk de term LET niet meer gebruiken. In het onderwijs wordt de term echter nog wel vaak aangetroffen, omdat het immers sterk de actie 'laat de variabele gelijk zijn aan een bepaalde waarde' benadrukt.

■ We mogen meer instructies op één regel plaatsen
In dit geval gebruiken we de dubbele punt als scheidingsteken. Het volgende voorbeeld geeft een treffende illustratie:

```
10 K=1000:R=8:W=K*(1+R/100):PRINT W:END
```

Probeer het maar eens uit!

■ We mogen standaardtermen afkorten
Voorbeeld:

In plaats van LIST mogen we ook schrijven:

```
L.
```

In plaats van PRINT mogen we ook schrijven:

```
PR.
```

Bovendien mag men PRINT door middel van een vraagteken aangeven. Voorbeeld:

```
10 K=100
20 ?K
30 END
```

Resultaat:

```
100
```

Ten slotte vermelden we dat afgekorte termen op het scherm bij het weergeven van listings altijd volledig worden weergegeven. Het vraagteken blijft in een listing wel onveranderd.

■ BYE

Als we het commando BYE geven, verschijnt op het scherm de volgende

afbeelding:

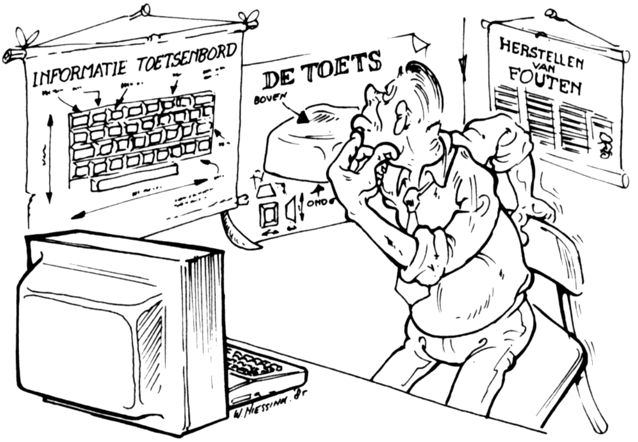
```
SELF TEST

MEMORY
AUDIO-VISUAL
KEYBOARD
ALL TESTS

SELECT, START or RESET
```

Door één of meer keren op de toets SELECT te drukken zien we een van de termen oplichten. De termen hebben betrekking op bepaalde tests: 'MEMORY' op het testen van het geheugen, 'AUDIO-VISUAL' op het testen van geluid en beeld, 'KEYBOARD' op het testen van het toetsenbord en ten slotte 'ALL TESTS' op het uitvoeren van alle tests.
Een betreffende test start men door op START te drukken. Een test kunnen we beëindigen door te drukken op:

RESET: de computer keert weer terug in de toestand waarin we met BASIC kunnen werken.

HELP: de computer keert weer terug naar het menu voor de eventuele keuze voor nog een test.

5. Herstellen van fouten en meer informatie over het toetsenbord



Inleiding

Het maken van fouten is een bij uitstek menselijke aangelegenheid. We merken dit met nadruk op omdat het nog weleens voorkomt dat een programma niet overeenkomstig onze verwachtingen werkt en welke uitspraak ligt dan meer voor de hand dan: 'De computer werkt niet goed!'. De ervaring leert dat er dan achteraf toch altijd weer een of meer fouten in het programma worden aangetroffen.
In de meeste gevallen zal de computer op dergelijke fouten attenderen door een of andere foutboodschap op het scherm te tonen. Een overzicht van alle foutboodschappen is in appendix 7 te vinden.
Fouten moeten uiteraard worden hersteld en in dit hoofdstuk zullen we tonen welke mogelijkheden de Atari hiertoe biedt. We gaan uit van een nieuw programma.
Dit programma luidt als volgt:

```
10 LET GULDENS=250
20 LET KOERS=0.9
30 LET MARKEN=KOERS*GULDENS
40 PRINT MARKEN
50 END
```

In dit voorbeeld hebben we gebruik gemaakt van de mogelijkheid om

langere namen voor variabelen te gebruiken. De winst is duidelijk: het programma is beter leesbaar. De Engelsen spreken weleens van 'self documentation'; het programma documenteert zich als het ware zelf.
Voor alle duidelijkheid geven we toch enige toelichting. Het programma berekent het aantal marken, gegeven het aantal guldens en de koers. Als gevolg van regel 10 zal de waarde 250 aan de variabele GULDENS worden toegekend. Als gevolg van regel 20 wordt aan KOERS de waarde 0.9 toegekend.
Hierna volgt de berekening van het overeenkomstig aantal marken en deze waarde wordt ten slotte afgedrukt.
We toetsen dit programma in. Voor de veiligheid geven we eerst het commando NEW, zodat we er zeker van zijn dat het geheugen werkelijk schoon is.

Tussenvoegen van een regel

Aan de hand van het zojuist ingevoerde programma bespreken we eerst het tussenvoegen van een regel.
Een dergelijke actie is noodzakelijk als we achteraf ontdekken dat er een instructie vergeten is. We kunnen een regel invoegen door eerst een regelnummer te kiezen dat ligt tussen de nummers van de regels waartussen de nieuwe regel moet komen.
Toets maar eens in:

```
15 PRINT GULDENS
25 PRINT KOERS
```

N.B. niet vergeten na het intoetsen van de tweede regel ook de RETURN-toets in te drukken.
Als we hierna het LIST-commando geven toont de computer ons:

```
10 LET GULDENS=250
15 PRINT GULDENS
20 LET KOERS=0.9
25 PRINT KOERS
30 LET MARKEN=KOERS*GULDENS
40 PRINT MARKEN
50 END
```

De ingetypte regels met de regelnummers 15 en 25 zijn, zoals we zien, op de juiste plaats ingevoegd. Ook het RUN-commando bevestigt dit. Na het commando RUN zien we:

```
250
0.9
225
```

Verwijderen van een regel

Stel dat we de zojuist tussengevoegde regels weer willen verwijderen. De procedure is eenvoudig: typ alleen de regelnummers van de te verwijderen regels in. We toetsen dus in:

15 *(gevolgd door de RETURN-toets)*
25 *(gevolgd door de RETURN-toets)*

Geven we hierna opnieuw het LIST-commando dan toont de computer ons:

```
10 LET GULDENS=250
20 LET KOERS=0.9
30 LET MARKEN=KOERS*GULDENS
40 PRINT MARKEN
50 END
```

De beoogde regels zijn inderdaad gewist.

Wijzigen van een regel

De Atari biedt ons verschillende mogelijkheden om een regel te wijzigen. Stel dat we als voorbeeld regel 20 willen wijzigen in:

```
20 LET KOERS=0.92
```

De eerste oplossing is een standaardoplossing die BASIC biedt: toets de te wijzigen regel gewoon opnieuw in, te beginnen met het regelnummer. We toetsen dus in:

```
20 LET KOERS=0.92
```

Geven we nu het LIST-commando dan toont de computer ons:

```
10 LET GULDENS=250
20 LET KOERS=0.92
30 LET MARKEN=KOERS*GULDENS
40 PRINT MARKEN
50 END
```

Nog wat bijzondere mogelijkheden

Onze computer is voorzien van een aantal bijzondere toetsen waardoor het wijzigen van programma's, zo men dat wenst, nog eenvoudiger is uit te voeren.
In het nu volgende bespreken we de toetsen één voor één:

De BACKSPACE-toets
Door op deze toets te drukken wissen we het laatst ingetoetste schrift-teken. Door deze toets meer keren in te drukken, dan wel deze toets

ingedrukt te houden, wissen we meer schrifttekens. Hierna kunnen we onmiddellijk weer andere schrifttekens intoetsen.

De cursor-besturingstoetsen ➡⬅⬇⬆
We kunnen een bepaald schriftteken in een regel eenvoudig verwisselen met behulp van de cursor-besturingstoetsen. De cursor is het oplichtende vierkantje. Drukken we met ingedrukte CONTROL-toets op de toets waar ➡ op staat, dan verplaatst de cursor zich naar rechts. Merk op dat het pijltje tegen de achtergrond van een wit vierkantje is afgebeeld. Bij het activeren van een toets die zo is afgebeeld moeten we altijd de CONTROL-toets ingedrukt houden. Dit 'toets-recept' geldt dus ook voor de overige cursor-besturingstoetsen.
Het recept om een bepaald schriftteken te vervangen is nu eenvoudig: stuur de cursor naar het betreffende schriftteken, toets het juiste schriftteken in en druk vervolgens op de RETURN-toets.

Overige combinaties met de CONTROL-toets
De CONTROL-toets biedt nog meer mogelijkheden. Als we bijvoorbeeld met ingedrukte CONTROL-toets op de toets met DELETE drukken, dan wordt daarmee een letter gewist en de tekst sluit automatisch aan. Deze handeling geven we trouwens aan met de notatie:

CTRL/DELETE

Overige mogelijkheden:

CTRL/INSERT	voor het invoegen van tekst.
CTRL/lettertoets	geeft grafisch symbool. De grafische mode kunnen we opheffen door met ingedrukte SHIFT-toets op de toets met CAPS te drukken.
CTRL/CLEAR	wist het scherm en plaatst de cursor in de linker bovenhoek.

Combinaties met de SHIFT-toets
Drukt men met ingedrukte SHIFT-toets op DELETE dan wordt de gehele regel verwijderd. We geven deze handeling aan met:

SHIFT/DELETE

Overige mogelijkheden:

SHIFT/INSERT	voor het invoegen van een regel.
SHIFT/CLEAR	als CTRL/CLEAR.
SHIFT/TAB	voor het aangeven van tabulatorstanden. Deze 'instel-mode' wordt met CTRL/TAB weer opgeheven.

CAPS
Door het indrukken van deze toets verkrijgen we kleine letters. Als we nogmaals op CAPS drukken verschijnen weer hoofdletters.

Als we op deze toets drukken, verschijnen letters in zogenaamd inverse-video-formaat. Dit houdt in dat wat eerst wit was nu zwart is en andersom dat wat eerst zwart was nu wit is. Drukken we nogmaals op deze toets dan wordt de oude situatie hersteld.

TAB
Wordt als tabulatortoets gebruikt.

BREAK
Met behulp van deze toets kunnen we een programma dat wordt 'gerund' onderbreken.

Afdrukken van een speciale toets
Als men probeert in te toetsen:

```
10 PRINT "SHIFT/CLEAR"
```

(N.B. Dus na " wordt met ingedrukte SHIFT-toets op CLEAR gedrukt.)

dan zal het scherm onmiddellijk na het indrukken van de CLEAR-toets worden gewist. Om dit effect te vermijden gebruiken we steeds de ESC-toets. Probeer maar:

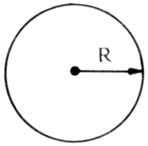
```
10 PRINT "ESC SHIFT/CLEAR"
```

De toetscombinatie SHIFT/CLEAR wordt nu door middel van het symbool weergegeven.

6. Over getallen en strings

Over verschillende getallen

In dit gedeelte zullen we onze aandacht allereerst vestigen op de verschillende soorten getallen die we in programma's mogen gebruiken.
We beschouwen in het nu volgende weer een eenvoudig programma. In dit programma wordt van een cirkel zowel de omtrek als het oppervlak berekend en wel als de straal R gegeven is.



We gebruiken hiervoor de bekende formules:

omtrek = 2 x 3.14159265 x R $\quad (= 2\pi R)$
oppervlak = 3.14159265 x R x R $\quad (= \pi R^2)$

Het programma voor een cirkel met straal R=10 luidt:

```
10 R=10
20 PI=3.14159265
30 OMTREK=2*PI*R
40 OPPERVLAK=PI*R*R
50 PRINT OMTREK
60 PRINT OPPERVLAK
70 END
```

Resultaat na het RUN-commando:

```
62.831853
314.159265
```

Het recept voor het opstellen van het programma is weer als vanouds: neem de formules over (regel 30 en 40) en ken de variabelen die bij de formules rechts van het teken = voorkomen van tevoren een waarde toe.

Wat opvalt is dat we het getal 3.14159265 van tevoren aan de variabele met de toepasselijke naam PI toekennen. Let wel, we kennen dit getal aan de variabele PI toe maar aangezien we PI voor geen enkele andere waarde zullen gebruiken, heeft PI hier de betekenis van 'een constante waarde'. Het was natuurlijk mogelijk om het getal, zowel in regel 30 als regel 40, direct uit te schrijven. De gekozen opzet is echter zonder meer overzichtelijker.
Als we het programma voor een andere waarde van R, bijvoorbeeld R=5, willen uitvoeren dan zullen we de eerste regel overeenkomstig moeten aanpassen:

```
10 R=5
```

We zullen trouwens straks zien dat de Atari ook mogelijkheden biedt om verschillende waarden aan variabelen toe te kennen zonder daarbij het programma te hoeven wijzigen.
Het voorbeeld toont zowel gehele als niet-gehele getallen. Voor alle duidelijkheid: een niet-geheel getal is een getal met een gedeelte achter de decimale punt. De gehele getallen ..., -4, -3, -2, -1, 0, 1, 2, 3, 4, ... worden ook wel eens met de term integer-getallen aangeduid. De niet-gehele getallen zoals 0.5 en 3.1415 worden weleens met de term reëele getallen aangeduid.
In een aantal gevallen willen we een niet-geheel getal in een geheel getal omzetten. In dit geval kan men gebruik maken van de INT-functie. Bij de INT-functie wordt het getal altijd naar beneden afgerond: om een getal echt af te ronden, dienen we er 0.5 bij op te tellen. Het volgende voorbeeld illustreert enkele gevallen:

```
10 LET A=3.7
20 LET B=INT(A)
30 LET C=INT(A+0.5)
40 PRINT "INT(3.7)=";B
50 PRINT "INT(3.7+0.5)=";C
60 END
```

Na het RUN-commando verschijnt:

```
INT(3.7)=3
INT(3.7+0.5)=4
```

Bij INT(3.7) wordt het getal 3.7 naar beneden afgerond en op deze wijze vinden we de waarde 3. Door bij de waarde 3.7 eerst 0.5 op te tellen, krijgen we als afgeronde waarde de meer voor de hand liggende waarde 4. Dit wordt aan de hand van regel 30 geïllustreerd. Uit het gegeven voorbeeld kunnen we tevens opmerken dat we kennelijk twee af te drukken zaken in een PRINT-instructie mogen opnemen en wel gescheiden door een puntkomma. In hoofdstuk 7 zullen we hier dieper op ingaan.
Naast de reeds genoemde getallen is er nog een categorie getallen die opvalt door een afwijkende notatievorm. Met deze notatievorm kunnen we met name zeer grote en zeer kleine waarden aangeven.
Beschouw als voorbeeld maar eens het volgende getal: 56 540 000 000 000 000. Dit is met recht een astronomisch groot getal... het zou de afstand tussen

één of ander hemellichaam en onze moeder aarde (bijvoorbeeld in kilometers) kunnen voorstellen. Een dergelijk getal kunnen we niet zomaar als getal intoetsen. Om dit probleem te ondervangen is er een notatie ingevoerd waarbij ieder getal kan worden voorgesteld als:

een getal x macht van tien

Ter herinnering:

macht van tien	**notatie**	**is gelijk aan**
10	10^1	10
10 x 10	10^2	100
10 x 10 x 10	10^3	1000
10 x 10 x 10 x 10	10^4	10000
...	...	...
10 x 10 ... 10	10^n	1000...

Voor het voorbeeld geldt, dat we het oorspronkelijke getal kunnen schrijven als:

5654 x 10 000 000 000 000

of:

5654×10^{13}

Het eerste getal noemt men de mantisse en het tweede getal noemt men de exponent. Nu dienen getallen bij het programmeren altijd op één regel te worden geschreven; het getal 13 dat, zoals getoond, rechts boven de tien wordt genoteerd dient in BASIC dan ook anders te worden geschreven. De afspraak is eenvoudig:

> Noteer eerst de mantisse, dan de letter E en ten slotte de macht van 10 (met andere woorden de exponent).

Dus voor het voorbeeld:

5654E13 (of 5654E+13)

Hetzelfde voorbeeld hanteert men bij de notatie van zeer kleine getallen. Bijvoorbeeld:

0.000 000 000 565 4

is gelijk aan:

5654×10^{-13}

en wordt dus geschreven als:

5654E-13

Het is ten slotte aardig om te weten dat we getallen met een willekeurig aantal cijfers kunnen invoeren: alleen de 10 meest betekenisvolle cijfers zullen in het geheugen worden onthouden. Een en ander heeft tot gevolg dat onze computer zeer nauwkeurig werkt. Het grootste getal dat onze computer nog kan bevatten is:

9.99999999E+97

Uiteraard brengt dergelijke kennis ons in verleiding om te zien wat er gebeurt als we nog grotere getallen krijgen, bijvoorbeeld door bovengenoemd getal met zichzelf te vermenigvuldigen. De computer toont dan onverbiddelijk:

```
ERROR- 11
```

In appendix 7 vinden we dat foutmelding 11 betekent:

`FLOATING POINT OVERFLOW` *(getal te groot)*

Inderdaad... u maakt het ook wel dol!

Variabelen waaraan we reeksen schrifttekens kunnen toekennen

Wat is het leven zonder letters. Geen boeken, geen kranten, geen tijdschriften, geen gebruiksaanwijzing voor de Atari.
Het hoeft ons dus niet te verbazen dat computers ook met letters en, meer in het algemeen, met schrifttekens allerlei handelingen kunnen uitvoeren.
Tekstverwerking bijvoorbeeld, vormt heden ten dage één van de belangrijkste toepassingsgebieden van de computer. Ook met de Atari kunnen we met schrifttekens allerlei handelingen uitvoeren.
Een reeks schrifttekens noemt men een string en een variabele waaraan men zo'n string kan toekennen noemt men heel toepasselijk een string-variabele. Dus:

> string = reeks schrifttekens
> string-variabele = variabele waaraan men een reeks schrifttekens kan toekennen.

Om onderscheid te kunnen maken tussen de zojuist behandelde variabelen die we nu maar voor het gemak getalvariabelen zullen noemen en de nu te behandelen string-variabelen, wordt achter de naam van de laatste categorie steeds een dollarteken geplaatst.

> Voor de naam van een string-variabele gelden de reeds behandelde regels, met dit verschil dat de naam moet eindigen met het dollarteken ($).

Voorbeelden van correcte namen zijn:

```
A$    A1$    EERSTE$    B3$
```

Wel dienen we met een zogenaamde DIM-instructie op te geven uit hoeveel tekens zo'n string mag bestaan. Voorbeeld:

```
10 DIM A$(5)
```

geeft aan dat aan A$ maximaal 5 schrifttekens mogen worden toegekend. Om aan te tonen dat we een string echt aan zo'n string-variabele kunnen toekennen, tonen we eerst het volgende voorbeeld:

```
10 DIM A$(5)
20 A$="ATARI"
30 PRINT A$
40 END
```

Resultaat:

```
ATARI
```

We zien hoe in regel 20 aan de string-variabele A$, de string "ATARI" wordt toegekend. Merk op dat zo'n string altijd tussen de bekende aanhalingstekens wordt geplaatst. Dit is gewoonweg een verplichting; er valt verder niets aan te begrijpen.
In plaats van:

```
A$="ATARI"
```

hadden we natuurlijk ook weer mogen schrijven:

```
LET A$="ATARI"
```

Eén en ander benadrukt ten slotte het feit dat het hier weer gewoon om een toekenningsinstructie gaat: aan een variabele wordt iets toegekend, in dit geval een string. De daaropvolgende regel:

```
PRINT A$
```

toont dat de 'waarde' (inhoud) van A$ dient te worden afgedrukt. Inderdaad, als resultaat verschijnt weer de oude bekende string. Merk echter op dat de aanhalingstekens zelf geen deel uitmaken van de string; deze worden immers niet afgebeeld.
In het nu volgende tonen we een aantal typische strings:

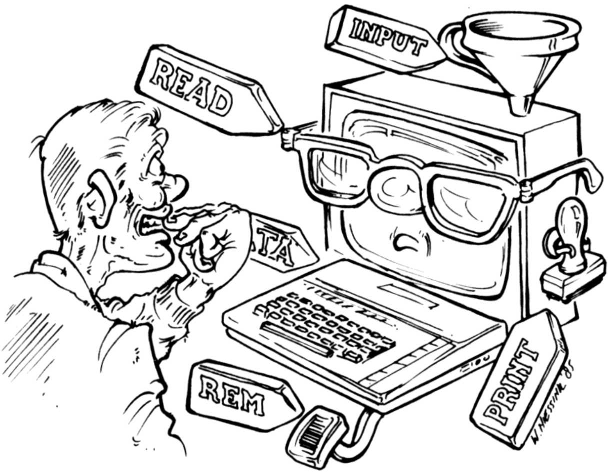
string	**commentaar**
"AB"	string, bestaande uit twee letters.
"A B"	string, bestaande uit drie tekens: twee letters en een spatie.
" "	string, bestaande uit drie tekens, en wel drie spaties.
"1254"	string, bestaande uit vier tekens, namelijk vier cijfers. Let wel, de cijfers vormen hier een string en niet een getal; ze staan ten slotte tussen de bekende aanhalingstekens. We zullen straks zien dat er wel mogelijkheden

string	commentaar
	zijn om dergelijke strings naar het 'gelijkluidende' getal om te zetten.
""	string, bestaande uit geen enkel schriftteken. We zouden deze wel zeer bijzondere string met het getal 0 bij de getallen kunnen vergelijken. We spreken hierbij dan ook wel eens van een nul-string (of lege string).

In hoofdstuk 13 zullen we tonen, wat voor functies men bij strings kan gebruiken. We zullen dan zien dat er veel mogelijkheden zijn die onder andere betrekking hebben op:

- het bepalen van zogenaamde substrings: dit zijn gedeelten van een oorspronkelijke string;
- het omzetten van schrifttekens in bepaalde codes en andersom, het omzetten van codes in schrifttekens.

7. INPUT, READ, DATA en RESTORE, PRINT en REM



De INPUT-instructie

Stel dat we een kapitaal KAP hebben, dat we tegen een rentepercentage van R procent gedurende N jaar laten uitstaan. De toekomstige waarde TW zal dan als volgt in een formule kunnen worden weergegeven:

$$TW=KAP \times (1+R \times 0.01)^N$$

De N rechts bovenaan geeft een zogenaamde macht aan: het is een verkorte schrijfwijze van:

$$\underbrace{(1+R \times 0.01) \times (1+R \times 0.01) \;\ldots\; \times (1+R \times 0.01)}_{N \text{ maal}}$$

De macht wordt bij de Atari met behulp van een dakje aangegeven. Zo luidt bovenstaande uitdrukking 'in BASIC':

```
TW=KAP*(1+R*0.01)^N
```

Stel dat KAP met *f* 1200,- overeenkomt en R met 8%. We kunnen dan met de computer TW berekenen; bijvoorbeeld voor N=2:

```
10 LET KAP=1200
20 LET R=8
30 LET N=2
40 LET TW=KAP*(1+R*0.01)^N
50 PRINT TW
60 END
```

Het recept is weer als vanouds: neem de formule over (regel 40) en zorg er voor dat aan alle grootheden die rechts van het =-teken voorkomen van tevoren een waarde is toegekend. Na de berekening voeren we met behulp van een PRINT-instructie de berekende waarde uit (regel 50).
Toets het programma maar eens in. Na het RUN-commando verschijnt:

```
1399.68
```

Dit is nauwelijks een algemeen programma te noemen. Het berekent namelijk de toekomstige waarde TW voor maar precies één bepaalde verzameling gegevens! Het zou plezierig zijn als de gebruiker op eenvoudige wijze de waarde van KAP, R en N zou kunnen wijzigen, zonder daarbij het programma te wijzigen.
De Atari biedt deze mogelijkheid en wel in de vorm van de INPUT-instructie. Deze instructie heeft steeds de vorm:

INPUT naam van de variabele

Tijdens het uitvoeren van het programma introduceert zo'n INPUT-instructie altijd een tijdelijke onderbreking. Dit houdt in dat de computer stopt met het uitvoeren van het programma en wel tot het moment dat de gebruiker een getal heeft ingetoetst. De computer maakt de onderbreking kenbaar door een vraagteken op het beeldscherm te tonen, we zien:

```
? ■
```

Als we dit teken zien weten we dus dat de computer bij een INPUT-instructie gestopt is en dat er van ons verwacht wordt een waarde in te toetsen (gevolgd door de RETURN-toets). Deze waarde wordt dan aan de variabele toegekend die na de INPUT-instructie is aangegeven. De computer zet hierna het programma onmiddellijk weer voort. Het zal duidelijk zijn dat we de eerste regels van ons programma door INPUT-instructies gaan vervangen om het programma algemeen toepasbaar te maken. Zo kunnen we zowel KAP, R als N vanaf het toetsenbord tijdens de executie van het programma invoeren!

```
10 INPUT KAP
20 INPUT R
30 INPUT N
40 LET TW=KAP*(1+R*0.01)^N
50 PRINT TW
60 END
```

Voer deze wijzigingen maar eens uit. We testen dit programma uit door dezelfde waarden voor KAP, R en N in te voeren; we moeten dan immers

weer hetzelfde resultaat verkrijgen! Na het RUN-commando zien we:

? ■

Dit is dan het resultaat van de eerste INPUT-instructie. We moeten dus nu een waarde voor KAP intoetsen. Toets in:

1200 *(gevolgd door de RETURN-toets)*

Direct hierna zien we weer:

? ■

Dit is het gevolg van de tweede INPUT-instructie. We toetsen nu een waarde voor R in:

8 *(gevolgd door de RETURN-toets)*

Opnieuw zien we:

? ■

Nu toetsen we een waarde voor N in:

2 *(gevolgd door de RETURN-toets)*

Hierna toont de computer direct het verlangde antwoord:

1399.68

Het programma is nu inderdaad in waarde toegenomen. Na opnieuw het RUN-commando te geven kunnen we de berekening opnieuw uitvoeren, maar nu voor andere waarden.
Toch kleeft er nog een belangrijk bezwaar aan het programma zoals het nu is: het programma is niet gebruikersvriendelijk. De gebruiker moet immers over veel voorkennis beschikken. We moeten namelijk precies weten wanneer we de waarden voor KAP, R en N moeten invoeren. Het zou veel plezieriger zijn als er op het beeldscherm teksten zouden verschijnen, waaruit dan precies blijkt wat we moeten doen. Nu is het inderdaad mogelijk om door middel van PRINT-instructies teksten af te drukken; we zagen dit reeds in hoofdstuk 2. We brengen weer in herinnering dat alle tekst in een PRINT-instructie steeds tussen aanhalingstekens (") geplaatst moet worden.
In het onderstaande zien we de verbeterde versie van ons oorspronkelijke programma. De oorspronkelijke regels zijn voor de duidelijkheid ongewijzigd:

```
2 PRINT "RENTE-BEREKENING"
3 PRINT "DIT PROGRAMMA BEREKENT"
4 PRINT "DE TOEKOMSTIGE WAARDE"
5 PRINT "VAN EEN BEGINKAPITAAL"
6 PRINT "DAT TEGEN R% GEDURENDE"
```

```
 7 PRINT "N JAREN WORDT UITGEZET"
 9 PRINT "VOER BEGINKAPITAAL IN"
10 INPUT KAP
15 PRINT "VOER RENTE-PERCENTAGE IN"
20 INPUT R
25 PRINT "VOER AANTAL JAREN IN"
30 INPUT N
40 LET TW=KAP*(1+R*0.01)^N
45 PRINT "DE TOEKOMSTIGE WAARDE IS"
50 PRINT TW
60 END
```

Zo op het oog al een 'fors' programma. Als we nu het RUN-commando geven verschijnt:

```
RENTE-BEREKENING
DIT PROGRAMMA BEREKENT
DE TOEKOMSTIGE WAARDE
VAN EEN BEGINKAPITAAL
DAT TEGEN R% GEDURENDE
N JAREN WORDT UITGEZET
VOER BEGINKAPITAAL IN
? ■
```

De verbetering is overduidelijk. De gebruiker hoeft nu niets van het programma te weten. De tekst geeft aan waar het programma betrekking op heeft en bovendien wordt keurig aangegeven wat voor waarden er moeten worden ingetoetst. We toetsen in:

1200 *(gevolgd door de RETURN-toets)*

Vervolgens verschijnt:

```
VOER RENTE-PERCENTAGE IN
? ■
```

Hierna toetsen we in:

8

en onmiddellijk hierna verschijnt:

```
VOER AANTAL JAREN IN
? ■
```

We toetsen in:

2

en als resultaat verschijnt:

```
DE TOEKOMSTIGE WAARDE IS
1399.68
```

Na dit voorbeeld zal de term 'gebruikersvriendelijk' ongetwijfeld veel duidelijker zijn.

Meer variabelen in een INPUT-instructie

We mogen meer variabelen in een INPUT-instructie opnemen. Het volgende voorbeeld illustreert dit:

```
10 INPUT A,B,C
20 D=A+B+C
30 PRINT D
40 END
```

Na het commando RUN toetsen we bijvoorbeeld in:

```
1,2,3
```

Kortom, drie getallen gescheiden door komma's. De computer toont dan:

```
6
```

We zien hoe na INPUT nu meer variabelen, gescheiden door komma's, zijn weergegeven.

READ, DATA en RESTORE

Er is nog een mogelijkheid om aan variabelen waarden toe te kennen. Deze mogelijkheid wordt gevormd door de zogenaamde READ/DATA-instructie. We geven onmiddellijk een voorbeeld:

```
10 PRINT "TOTAALWAARDE"
20 READ A,B,C,D,E,F
30 S=A+B+C+D+E+F
40 PRINT "DE BEREKENDE WAARDE IS"
50 PRINT S
60 DATA 8,7,9,8,6,10
70 END
```

Na het RUN-commando verschijnt:

```
TOTAALWAARDE
DE BEREKENDE WAARDE IS
48
```

De eerste regel geeft geen problemen. De tweede regel luidt:

```
READ A,B,C,D,E,F
```

Op grond van de term READ weet de computer dat de waarden van deze variabelen in een zogenaamde DATA-lijst, een lijst waarden na

de term DATA, zijn opgenomen. Zo zal aan A de waarde 8, aan B de waarde 7 enz. worden toegekend. Hierna voert de computer de regels 30, 40 en 50 uit. Regel 60 wordt gewoonweg overgeslagen en regel 70 geeft aan dat ook aan dit programma een eind komt.
Het voordeel van deze constructie boven de toekenning met behulp van toekenningsinstructies is dat er relatief weinig instructies nodig zijn en dat we maar één regel hoeven te wijzigen om het programma voor andere waarden uit te voeren (namelijk regel 30).
Er mogen trouwens meer READ- en DATA-instructies in een programma voorkomen. We geven twee voorbeelden.

Voorbeeld 1

```
...
20 READ A,B,C
25 READ D,E,F
...
70 DATA 8,7,9,8,6,10
...
```

Voorbeeld 2

```
...
20 READ A,B,C,D,E,F
...
70 DATA 8,7,9
75 DATA 8,6,10
...
```

Beide voorbeelden zijn in principe gelijkwaardig aan het reeds besproken voorbeeld.
Een eenmaal uitgelezen DATA-lijst kan opnieuw worden uitgelezen als we de RESTORE-instructie gebruiken. Een eenvoudig voorbeeld illustreert het gebruik van RESTORE:

```
10 READ A
20 PRINT A
30 RESTORE
40 READ B
50 PRINT B
60 DATA 3
70 END
```

Na het RUN-commando verschijnt:

```
3
3
```

Als gevolg van de RESTORE-instructie op regel 30 zal de DATA-lijst weer opnieuw worden uitgelezen. Zo wordt aan A én aan B de waarde 3 toegekend.

De PRINT-instructie

De PRINT-instructie is stellig een van de meest krachtige instructies van onze Atari.
In het voorgaande zagen we al verschillende voorbeelden, zoals het afdrukken van tekst en het afdrukken van waarden van variabelen. We kunnen deze mogelijkheden ook in één PRINT-instructie combineren. Het volgende programma illustreert dit:

```
10 A=3:B=4:C=A*B
20 PRINT "C=";C
30 END
```

De eerste regel bevat drie instructies. Regel 20 bevat een PRINT-instructie. Eerst zal de tekst C= worden afgedrukt en direct hierna volgt dan de waarde van C.
Zo verschijnt na het RUN-commando uiteindelijk de volgende tekst:

```
C=12
```

Nog een voorbeeld:

```
10 DIM A$(3)
20 DIM B$(4)
30 A$="WAL":B$="NOOT"
40 PRINT A$;B$
50 END
```

Na het RUN-commando verschijnt:

```
WALNOOT
```

Het geheim schuilt in het teken ; (puntkomma). Dit teken geeft aan dat een volgende af te drukken zaak direct achter de vorige moet worden geplaatst (voor een getal wordt dan altijd een spatie geplaatst).
Maken we gebruik van een komma als scheidingsteken dan zal de Atari automatisch een indeling in vier kolommen aanhouden. Voorbeeld:

```
10 PRINT "KOLOM1","KOLOM2","KOLOM3","KOLOM4"
20 PRINT 1,2,3,4
30 PRINT 5,6,7,8
40 END
```

Na het RUN-commando zien we:

```
KOLOM1     KOLOM2     KOLOM3     KOLOM4
 1          2          3          4
 5          6          7          8
```

Uit dit voorbeeld kunnen we tevens opmaken dat we ook meer dan twee af te drukken zaken in een PRINT-instructie mogen opnemen.
Wat zijn zoal de 'zaken' die we in een PRINT-instructie mogen opnemen?

Welnu, samengevat zijn dit:
- teksten geplaatst tussen aanhalingstekens (");
- variabelen;
- uitdrukkingen zoals A*B, waarvan ten slotte de uitkomst wordt afgebeeld.

Als scheidingstekens mogen we gebruiken:
- de puntkomma (;), de volgende af te drukken zaak wordt direct achter het voorgaande afgebeeld;
- de komma (,), er wordt automatisch een kolomindeling aangehouden.

De scheidingstekens mogen ook als laatste teken in een PRINT-instructie worden gebruikt. Het volgende voorbeeld illustreert alle mogelijkheden:

```
10 PRINT "RATJETOE"
20 DIM C$(5)
30 A=1:B=2:C$="GETAL"
40 PRINT A,B;C$
50 PRINT A+B;C$;
60 PRINT C$
70 PRINT "GEHEEL";C$
80 PRINT
90 PRINT "EINDE"
100 END
```

Na het RUN-commando verschijnt:

```
RATJETOE
1          2GETAL
3GETALGETAL
GEHEELGETAL

EINDE
```

We laten het aan de lezer zelf over om deze wartaal te ontcijferen. We vermelden slechts dat een PRINT-instructie zonder meer (zie regel 80) tot gevolg heeft dat een regel wordt overgeslagen.

REM-instructie

Het laatste onderdeel van dit hoofdstuk vormt de REM-instructie. Met behulp van deze instructie kunnen we ons programma van commentaar voorzien. De term REM komt van het Engelse REMark, hetgeen zoveel als 'opmerking' wil zeggen. Alle REM-instructies zullen tijdens het uitvoeren van het programma gewoon worden overgeslagen, met andere woorden het opgenomen commentaar wordt niet afgedrukt. Het volgende programma illustreert dit:

```
10 REM ILLUSTRATIE VAN
20 REM DE REM-INSTRUCTIE
30 PRINT "DIT WORDT AFGEDRUKT"
40 END
```

Na het RUN-commando verschijnt:

```
DIT WORDT AFGEDRUKT
```

We zien hoe in regel 10 en 20 REM-instructies zijn opgenomen en dat het daar geplaatste commentaar niet wordt afgedrukt. Plaatsen we meerdere instructies op een regel (gescheiden door een dubbele punt), dan zien we dat alle instructies na REM worden overgeslagen. Voorbeeld:

```
10 PRINT "DIT WORDT AFGEDRUKT":REM:PRINT "DIT OOK":
PRINT "EN DIT"
20 PRINT "EINDE"
30 END
```

Na het RUN-commando zien we:

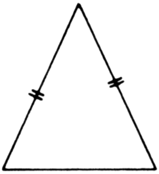
```
DIT WORDT AFGEDRUKT
EINDE
```

De instructies PRINT "DIT OOK" en PRINT "EN DIT" op regel 10 worden dus ook overgeslagen!

8. Besturingsinstructies

De IF...THEN-instructie en de GOTO-instructie

In de vorige programma's hadden we steeds te maken met problemen waarbij de oplossing werd verkregen door een serie instructies één voor één, van het begin tot het einde uit te voeren. Stel nu eens dat we in een programma een drietal gegevens opgeven die dan betrekking hebben op de lengtes van de drie zijden van een driehoek. Met het beschikbare arsenaal aan instructies is het moeilijk om, op grond van de lengtes van die zijden uit te maken of we nu al of niet met een gelijkbenige driehoek te maken hebben.
Pro memorie: een gelijkbenige driehoek is een driehoek waarvan tenminste twee zijden aan elkaar gelijk zijn:



Het ligt voor de hand om met betrekking tot ons probleem vragen te construeren in de trant van:

is de eerste zijde gelijk aan de tweede?

en zo niet:

is de eerste zijde gelijk aan de derde?

en zo niet:

is de tweede zijde gelijk aan de derde?

Kortom, er zijn instructies nodig waarmee vragen als bovenstaande uitdrukkingen kunnen worden aangegeven. BASIC biedt hiervoor een eenvoudige instructie die deze mogelijkheid biedt, namelijk de IF...THEN-instructie. Deze instructie heeft steeds de volgende vorm:

IF zekere voorwaarde THEN instructie

Voorbeeld:

```
IF A=3 THEN PRINT "A IS 3"
```

Deze uitdrukking wil in woorden zeggen:

Als de waarde van de variabele A gelijk is aan 3 voer dan de instructie uit die na THEN staat; kortom, de PRINT-instructie wordt dan uitgevoerd.
Als aan de omschreven voorwaarde (A=3) niet wordt voldaan, wordt het programma voortgezet met de instructie die staat op de regel na de regel waar de IF-instructie voorkomt.

We zien hoe tussen de termen IF en THEN (vertaald: ALS en DOE DAN) een voorwaarde staat aangegeven die, op het moment dat de computer die IF...THEN-instructie uitvoert, al of niet opgaat.
Na THEN mag ook een sprong naar een regelnummer worden aangegeven, bijvoorbeeld:

```
IF A=3 THEN GOTO 140
```

In dit geval zal een sprong naar regelnummer 140 worden uitgevoerd alwaar het programma dan wordt voortgezet; dit alles dan onder de voorwaarde dat A gelijk is aan 3.
De uitdrukking GOTO na THEN mag eventueel ook worden weggelaten. Op deze wijze wordt bovenstaande uitdrukking:

```
IF A=3 THEN 140
```

Het is overigens aardig om te weten dat dit de enige vorm is die in het 'oer-BASIC' van de oorspronkelijke ontwerpers Kemeny en Kurz is toegestaan.
Een instructie als 'GOTO regelnummer', bijvoorbeeld:

```
GOTO 140
```

mag ook als afzonderlijke instructie op een regel verschijnen. In dit geval

zal *altijd* (men spreekt ook wel van *onvoorwaardelijk*) de sprong naar het aangegeven regelnummer worden uitgevoerd. De IF...THEN-instructie met regelnummer voert eigenlijk een voorwaardelijke sponginstructie uit.

In het nu volgende zullen we eerst een oplossing van ons driehoeks-probleem laten zien, waarbij alleen maar regelnummers achter THEN worden toegelaten. Ongetwijfeld zal opvallen hoe gekunsteld een dergelijke oplossing wel wordt.
Allereerst kunnen we de drie lengtes van de zijden, via INPUT-instructies, toekennen aan de variabelen:

Z1   Z2   en   Z3

(N.B. Geen spatie tussen nummer en Z plaatsen; dat is bij het aangeven van namen, zoals bekend, niet toegestaan.)
Vervolgens zouden we in het programma twee PRINT-instructies kunnen opnemen. De eerste PRINT-instructie geeft aan dat de driehoek niet gelijkbenig is:

```
PRINT "NIET GELIJKBENIG"
```

en de tweede dat dit juist wel het geval is:

```
PRINT "GELIJKBENIG"
```

Vóór deze instructies staan altijd regelnummers die we nu maar, voor het gemak, met a en b zullen aangeven. Dus:

```
a PRINT "NIET GELIJKBENIG"
b PRINT "GELIJKBENIG"
```

Voor alle duidelijkheid: de regelnummers a en b dienen straks uiteraard getallen voor te stellen. We zullen zo dadelijk zien welke getallen.
Om na te gaan of een van de zijden gelijk is aan een andere, nemen we dan de volgende drie IF...THEN-instructies op:

```
... IF Z1=Z2 THEN b
... IF Z1=Z3 THEN b
... IF Z2=Z3 THEN b
```

Kortom, hier worden alle mogelijkheden nagegaan en steeds geldt dat als aan één van die mogelijkheden wordt voldaan, een sprong naar regelnummer b volgt alwaar de betreffende PRINT-instructie staat:

```
b PRINT "GELIJKBENIG"
```

We kunnen tevens constateren dat, als de computer de drie IF...THEN-instructies heeft afgewerkt zonder dat aan één van de voorwaarden werd voldaan, de driehoek kennelijk geen twee gelijke zijden had. Het ligt dus voor de hand om onmiddellijk na deze serie IF...THEN-instructies de instructie:

```
a PRINT "NIET GELIJKBENIG"
```

op te nemen. We krijgen dus een volgende constructie:

```
... IF Z1=Z2 THEN b
... IF Z1=Z3 THEN b
... IF Z2=Z3 THEN b
  a PRINT "NIET GELIJKBENIG"
... ...
  b PRINT "GELIJKBENIG"
```

Het programma begint al vorm te krijgen. Er is nog één duidelijk probleem op te lossen en wel het volgende. Als het programma heeft geconstateerd dat de driehoek niet gelijkbenig is, kortom de PRINT-instructie op regel a is uitgevoerd, zal de PRINT-instructie op regel b niet meer mogen worden uitgevoerd. Het zou immers vreemd zijn als we eerst de mededeling krijgen 'NIET GELIJKBENIG' om even daarna te vernemen 'GELIJKBENIG'... We moeten dus vermijden dat regel b kan worden uitgevoerd als regel a is uitgevoerd. Een van de manieren om dat te bereiken is, om na regel a een zogenaamde STOP-instructie te plaatsen, dus:

```
  a PRINT "NIET GELIJKBENIG"
... STOP
```

Zo, we hebben nu de volledige constructie doorgenomen. Het volledige programma luidt:

```
10 REM ANALYSE OF EEN DRIEHOEK
20 REM GELIJKBENIG IS
30 PRINT "ZIJDE1=";:INPUT Z1
40 PRINT "ZIJDE2=";:INPUT Z2
50 PRINT "ZIJDE3=";:INPUT Z3
60 IF Z1=Z2 THEN 110
70 IF Z1=Z3 THEN 110
80 IF Z2=Z3 THEN 110
90 PRINT "NIET GELIJKBENIG"
100 STOP
110 PRINT "GELIJKBENIG"
120 END
```

Voorbeeld van een resultaat:

```
ZIJDE1=?4
ZIJDE2=?6
ZIJDE3=?4
GELIJKBENIG
```

De gekozen oplossing doet inderdaad zeer gekunsteld aan. We tonen nu een oplossing waarbij we tevens gebruik maken van het feit dat na THEN eventueel meer instructies mogen worden opgenomen:

```
30 PRINT "ZIJDE1=";:INPUT Z1
40 PRINT "ZIJDE2=";:INPUT Z2
50 PRINT "ZIJDE3=";:INPUT Z3
```

```
60 IF Z1=Z2 THEN PRINT "GELIJKBENIG":GOTO 100
70 IF Z1=Z3 THEN PRINT "GELIJKBENIG":GOTO 100
80 IF Z2=Z3 THEN PRINT "GELIJKBENIG":GOTO 100
90 PRINT "NIET GELIJKBENIG"
100 END
```

Hier wordt iedere keer een test uitgevoerd en als de voorwaarde wordt bevestigd volgt direct de boodschap GELIJKBENIG en volgt een sprong naar het einde van het programma.
Het belangrijkste van deze oplossing is wel dat aan onze natuurlijke denkmethode recht wordt gedaan, kortom de taal vormt hier geen belemmering om de oplossing op te schrijven die in feite direct voor de hand ligt.

Relatietekens

In het programma worden bij de IF...THEN-instructies steeds twee waarden van variabelen met elkaar vergeleken en wel met het teken = ('is gelijkteken'). Hier wil het teken = inderdaad zeggen 'is gelijk aan'.
Men kan ook vragen construeren als 'is ZIJDE1 groter dan ZIJDE2' of 'is ZIJDE1 kleiner dan ZIJDE2'. In dit geval worden respectievelijk de tekens > en < gebruikt. Dergelijke tekens noemen we relatietekens: ze brengen twee grootheden met elkaar in relatie. In de nu volgende tabel zien we een overzicht van al de toegestane relatietekens:

teken	betekenis	voorbeeld
=	is gelijk aan	IF A=B THEN ...
<	kleiner dan	IF A<B THEN ...
<=	kleiner dan of gelijk aan	IF A<=B THEN ...
>	groter dan	IF A>B THEN ...
>=	groter dan of gelijk aan	IF A>=B THEN ...
<>	ongelijk aan	IF A<>B THEN ...

De termen AND, OR en NOT

We mogen ook voorwaarden combineren. Denk als voorbeeld maar eens aan de voorwaarden:

L>16 en L<30

Deze twee voorwaarden mogen worden gecombineerd met de termen AND en OR tot:

L>16 AND L<30

en

L>16 OR L<30

De term AND in bovengenoemde uitdrukking geeft dan aan, dat aan

de combinatievoorwaarde wordt voldaan, als aan beide deelvoorwaarden links en rechts van de term AND wordt voldaan. Zo voldoet de waarde L=20, immers 20>16 en tevens 20<30.
De term OR in bovengenoemde uitdrukking geeft aan, dat aan deze combinatievoorwaarde wordt voldaan als tenminste aan één van de deelvoorwaarden wordt voldaan. Bij het getoonde voorbeeld is dit trouwens voor iedere waarde van L het geval.
In tabelvorm:

deelvoorwaarde 1 (bijv. L>16)	**deelvoorwaarde 2** (bijv. L<30)	**combinatie met AND** (bijv. L>16 AND L<30)
juist	juist	juist
onjuist	onjuist	onjuist
onjuist	juist	onjuist
juist	onjuist	onjuist

en evenzo:

deelvoorwaarde 1 (bijv. L>16)	**deelvoorwaarde 2** (bijv. L<30)	**combinatie met OR** (bijv. L>16 OR L<30)
juist	juist	juist
onjuist	onjuist	onjuist
onjuist	juist	juist
juist	onjuist	juist

We mogen zelfs meer voorwaarden combineren, bijvoorbeeld:

```
IF (L>16 AND L<30) OR L>65 THEN . . .
```

Merk op hoe hier haakjes worden gebruikt om eventuele misverstanden te voorkomen. Bovenstaande voorwaarde gaat op als L tussen 16 en 30 ligt, dan wel als L groter is dan 65.
Ten slotte mogen we nog de term NOT gebruiken. NOT draait de zaak gewoon om... IF NOT L<16 THEN ... heeft tot gevolg dat de instructie na THEN wordt uitgevoerd als L juist niet kleiner is dan 16 (met andere woorden L>=16).

Tijd voor een avontuur!

Met de tot nu toe besproken instructies kunnen we al boeiende spelletjes maken. Laten we de daad maar eens bij het woord voegen... Geachte lezer, in het nu volgende nemen we u mee in een boeiend avontuur. Adventure games, zoals de Engelse benaming van dit type spellen luidt, vormen stellig een van de meest gespeelde computerspellen van nu.
Bij een 'adventure-spel' worden we steeds in een bepaalde situatie gedwongen waar we dan door inventiviteit maar weer uit moeten zien te komen.

De omgeving waarin het spel zich afspeelt, speelt daarbij uiteraard een sterk tot de verbeelding sprekende rol.
Het nu volgende adventure-spel zullen we eerst demonstreren aan de hand van een tekst zoals die op het scherm kan verschijnen. Daarna zal worden getoond hoe dit gehele spel geprogrammeerd is. Op deze wijze

zien we ook eens hoe een wat groter programma tot stand komt. We starten nu met een kleine tekst zoals deze zou kunnen verschijnen:

```
EINDELIJK LAND IN ZICHT... NOG EEN PAAR METER
ZWEMMEN
JA HOOR VASTE GROND ONDER DE VOETEN
IK BEVIND ME NU OP EEN STRAND
ER LIGGEN NOG EEN AANTAL BEENDEREN...
VERSCHRIKKELIJK!
ACHTER HET STRAND LIGGEN GROTE BERGEN
IK KAN NU LANGS HET STRAND GAAN (1) OF
OVER DE BERGEN GAAN (2)
TYPE 1 OF 2 IN
? 2
WE MAKEN EEN BARRE TOCHT OVER DE BERGEN
WE ZIEN NU EEN WEG EN KUNNEN LINKS (1)
DAN WEL RECHTS AF (2)
? 2
WE VERVOLGEN DEZE WEG...
DE WEG GAAT OVER IN EEN MOERAS
LANGZAAM ZAKKEN WE IN HET MOERAS...
```

```
HELAAS HET AVONTUUR HEEFT VOOR U EEN
SLECHTE WENDING GENOMEN
U KUNT HET NATUURLIJK NOG EEN KEER
PROBEREN DOOR HET RUN-COMMANDO TE GEVEN
EINDE SPEL
```

De tekst geeft duidelijk aan om wat voor soort spel het gaat. U wordt voortdurend geconfronteerd met een bepaalde situatie en een beperkt aantal keuzemogelijkheden. In dit geval gaat het om een duidelijke tekst-adventure. Er zijn ook adventures waarbij fraaie plaatjes worden getoond die ons dan nog suggestiever aangeven in wat voor benarde situatie we ons bevinden.
In dit geval geven we onze keuze steeds aan door een 1 of 2 in te typen. Bij de meer uitgebreide adventures kan men ook woorden, ja zelfs kleine zinnetjes intypen. Het principe is echter steeds gelijk, we zullen daar nu wat dieper op ingaan.

Programma

In het nu volgende zullen we zien hoe het getoonde adventure-spel uiteindelijk tot stand is gekomen. We zijn begonnen met een kaart te maken van de omgeving waarin zich deze adventure afspeelt. Vaak wordt zo'n kaart ook (voor een deel) aan de speler getoond, uiteraard om een en ander meer aanschouwelijk te maken. We zijn voor ons spel uitgegaan van een eiland met bergen, een rivier en paden. Op dit eiland is dan een schat te vinden. In feite vormt het vinden van die schat het uiteindelijke doel van het spel.
De in stijl gehouden tekening op de volgende pagina toont de kaart waarvan we zijn uitgegaan.
De namen zijn natuurlijk zelf verzonnen. De tekening geeft al een aardig idee van de mogelijke situaties waarin we zoal terecht kunnen komen. We gaan er bij dit spel van uit dat we in het begin altijd op het verlaten strand aankomen. Vanaf dit punt kunnen we dan over de bergen gaan dan wel een pad volgen. Als we over de bergen gaan komen we altijd, zoals aangegeven, op een pad uit dat naar een tweesprong leidt. De linkertak eindigt in een ravijn en de rechtertak in een moeras. Als we echter na het strand direct het pad volgen komen we bij een rivier uit, die we eventueel kunnen oversteken. We kunnen natuurlijk ook die rivier volgen. De rivier leidt allereerst naar een grot met een verborgen zwaard. Om het spel tot een goed einde te brengen moeten we nu met het zwaard terugkeren en wel tot het pad voor de rivier. Volgen we echter de rivier en wel tot aan een bron dan zal er iets 'naars' gebeuren...
Als we met het zwaard zijn teruggekeerd tot voor de rivier vervolgen we onze weg, over de rivier, langs het pad dat eerst langs de zee loopt. Bij een tweesprong moeten we voor een succesvol verloop van het spel rechtsaf gaan, kortom we moeten naar de burcht gaan. Hier moeten we dan eerst door een poort waar een enge draak ons opwacht. Geen nood, met het zwaard is de klus snel geklaard. Hierna zien we twee deuren; achter een van de deuren ligt de schat.
Als we bij de tweesprong niet rechtsaf zijn gegaan komen we weer bij de bergen uit, die we dan eventueel weer kunnen beklimmen. In dat geval

VERLATEN
STRAND
BRON
?
DE BERGEN
VAN
OUWE BEN
ZWAARD
GROT
RIVIER "DE ALAZONE"
HET MOERAS
VAN
TWOZZLE
DE
BURCHT
PAD
PAD
DE BERGEN
VAN
JONGE BEN
HET EILAND
BARARABAS
XMDII

komen we natuurlijk weer bij het strand uit zodat we opnieuw kunnen beginnen...
Let wel, deze kaart en deze beschrijving zijn van tevoren door de programmeur gemaakt. Op deze wijze heeft de programmeur eigenlijk het volledige spel in zijn hoofd en kan het werkelijke programmeren beginnen. Het zal duidelijk zijn dat het programma uit veel PRINT-instructies bestaat en tevens uit veel IF...THEN-instructies. Bij de eerste opzet van het programma gebruiken we geen regelnummers maar symbolische labels waar naar toe kan worden gesprongen. Om deze werkwijze te demonstreren geven we eerst een voorbeeld:

```
10 PRINT "WE ZIJN NU OP HET STRAND"
20 PRINT "U GAAT NU VIA HET PAD VERDER"
30 PRINT "U KUNT NU TERUGGAAN (1) DAN WEL DOORGAAN (2)
40 INPUT KEUZE
50 IF KEUZE=1 THEN GOTO 10
60 PRINT "GOED WE GAAN DUS DOOR"
```

Dit programmagedeelte vervangen we nu door het volgende gedeelte, waarin *geen* regelnummers maar wel *symbolische labels* zijn gebruikt.

strand — PRINT we zijn op het strand en vervolgen de weg langs een pad en kunnen daarna teruggaan (1) of doorgaan (2).
IF keuze=1 THEN strand
enz.

Merk op dat in dit voorbeeld strand een bepaalde positie in het programma aangeeft, waar we al of niet naar toe kunnen springen.

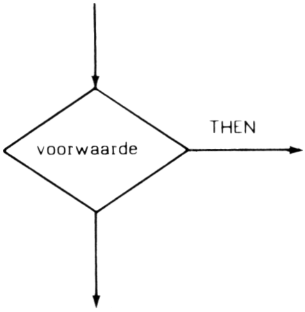
Na deze uitleg volgt hier de ruwe opzet van het adventure-programma:

strand — PRINT we zijn op strand; keuze langs strand (1) of over bergen (2).
IF keuze=2 THEN over bergen
voor rivier — PRINT komen bij rivier; keuze rivier volgen (1) of over rivier zwemmen (2).
IF keuze=2 THEN over rivier
PRINT vervolgen rivier; keuze weg vervolgen (1) of grot in gaan (2).
IF keuze=2 THEN in grot
PRINT we vervolgen rivier, komen bij bron; keuze drinken (1) dan wel over de bergen gaan (2).
IF keuze=2 THEN strand
PRINT drankje niet goed: GOTO eind

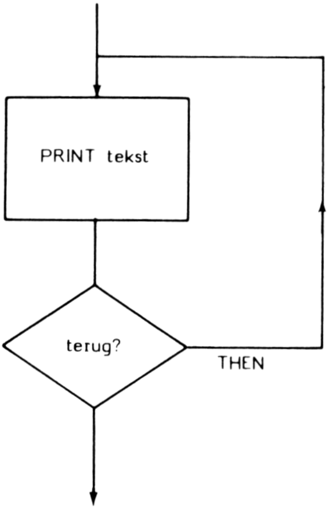
over bergen — PRINT tocht over bergen komt bij pad uit en een tweesprong; keuze links (1) of rechts (2).
IF keuze=1 THEN vallen in gat: GOTO eind
PRINT vervolgen onze weg, komt in moeras uit, we zakken in moeras weg...
GOTO eind

```
over rivier     PRINT tocht langs pad, komen bij tweesprong uit; keuze
                      links (1) of rechts (2).
                IF keuze=2 THEN oud kasteel
                PRINT vervolgen onze weg, komen bij een tweesprong;
                      keuze rechts (1) of links (2).
                IF keuze=2 THEN vervolg weg
                PRINT we vervolgen de weg; keuze weg vervolgen (2) of
                      over de bergen gaan (1).
                IF keuze=1 THEN strand
                GOTO over de bergen bij het punt 'tweesprong'

in grot         PRINT in grot ligt zwaard; keuze pakken (1) of laten
                      liggen (2).
                IF keuze=1 THEN maak variabele A$ gelijk aan 'ZWAARD'
                PRINT we kunnen nu terug (1) dan wel in een kreekje
                      springen (2).
                IF keuze=2 THEN water te koud...: GOTO eind
                PRINT we kunnen nog verder terug (1) dan wel weg weer
                      vervolgen (2).
                IF keuze=1 THEN voor rivier
                PRINT we worden betoverd...
                GOTO eind

oud kasteel     PRINT we zien een kasteel; keuze door poort gaan (1)
                      of door slotgracht (2).
                IF keuze=2 THEN verdrinken: GOTO eind
                PRINT komen op binnenplaats met draak
                IF we het zwaard niet hebben THEN dood: GOTO eind
                PRINT komen voor 2 deuren; keuze linker (1) of rech-
                      ter (2).
                IF keuze=2 THEN vallen in diepte: GOTO eind
                PRINT we vinden de schat
```

Zie zo... dat was een heel verhaal dat ons in ieder geval demonstreert hoe we tot een groter programma komen. In dit geval hebben we geen schema-technieken gebruikt. Dat kan echter heel raadzaam zijn. De IF...THEN-instructie wordt dan aangegeven met:



Bijvoorbeeld:



We zien hier een eenvoudig voorbeeld waar al of niet naar een bepaald gedeelte wordt teruggekeerd. Dit wordt in feite treffend geïllustreerd omdat alle te volgen wegen in het programma ook werkelijk grafisch zijn aangegeven. Als oefening zou u het zojuist gegeven spel eens in een dergelijk 'stroomdiagram' kunnen omzetten.

Ten slotte tonen we het programma dat aan de hand van onze ruwe opzet uiteindelijk is verkregen:

```
10 CLR:PRINT "EINDELIJK LAND IN ZICHT... NOG EEN
PAAR METER ZWEMMEN"
20 PRINT "JA HOOR VASTE GROND ONDER DE VOETEN"
30 PRINT "IK BEVIND ME NU OP EEN STRAND"
40 PRINT "ER LIGGEN NOG EEN AANTAL BEENDEREN...
VERSCHRIKKELIJK!"
50 PRINT "ACHTER HET STRAND LIGGEN GROTE BERGEN"
60 PRINT "IK KAN NU LANGS HET STRAND GAAN (1) OF
OVER DE BERGEN GAAN (2)"
70 PRINT "TYPE 1 OF 2 IN"
75 INPUT KEUZE
80 IF KEUZE=2 THEN GOTO 300
90 PRINT "IK LOOP NU LANGS HET STRAND"
100 PRINT "LINKS VAN MIJ BEVINDEN ZICH DE BERGEN"
110 PRINT "VOOR ME LIGT EEN RIVIER"
120 PRINT "IK KAN NU LANGS DE RIVIER GAAN (1) OF
ER OVERHEEN ZWEMMEN (2)"
125 INPUT KEUZE
130 IF KEUZE=2 THEN GOTO 600
```

```
140 PRINT "WE VERVOLGEN ONZE WEG LANGS DE RIVIER"
150 PRINT "LINKS VAN ONS ZIEN WE EEN GROT"
160 PRINT "WE KUNNEN NU ONZE WEG VERVOLGEN (1) DAN
WEL DE GROT IN GAAN (2)"
165 INPUT KEUZE
170 IF KEUZE=2 THEN GOTO 900
180 PRINT "WE VERVOLGEN ONZE WEG LANGS DE RIVIER"
190 PRINT "IN DE VERTE HOREN WE HET SPRANKELENDE
GELUID VAN EEN BRON"
200 PRINT "WE BEREIKEN NU DE BRON ACHTER DE BRON
LIGGEN BERGEN"
210 PRINT "WE KUNNEN NU BRONWATER DRINKEN (1) DAN
WEL OVER DE BERGEN GAAN (2)"
215 INPUT KEUZE
220 IF KEUZE=2 THEN GOTO 30
230 PRINT "DIT DRANKJE IS NIET GOED..."
240 PRINT "U VALT DOOD NEER..."
250 GOTO 2000
300 REM OVER DE BERGEN
310 PRINT "WE MAKEN EEN BARRE TOCHT OVER DE BERGEN"
320 PRINT "WE ZIEN NU EEN WEG EN KUNNEN LINKS (1)
DAN WEL RECHTSAF (2)"
325 INPUT KEUZE
330 IF KEUZE=1 THEN PRINT "WE VALLEN PLOTSELING
IN EEN DIEP GAT":GOTO 2000
340 PRINT "WE VERVOLGEN DEZE WEG..."
350 PRINT "DE WEG GAAT OVER IN EEN MOERAS"
360 PRINT "LANGZAAM ZAKKEN WE IN HET MOERAS..."
370 GOTO 2000
600 REM OVER DE RIVIER
610 PRINT "WE MAKEN EEN LANGE TOCHT"
620 PRINT "EERST ZIEN WE EEN BOS LINKS VAN ONS"
630 PRINT "NU BEREIKEN WE EEN TWEESPRONG WE KUNNEN
LINKS (1) OF RECHTS (2)"
635 INPUT KEUZE
640 IF KEUZE=2 THEN GOTO 1200
650 PRINT "WE VERVOLGEN ONZE WEG. LINKS ZIEN WE
NU BERGEN"
660 PRINT "WE KUNNEN NU OVER DE BERGEN (1) DAN WEL
ONZE WEG VERVOLGEN (2)"
665 INPUT KEUZE
670 IF KEUZE=1 THEN GOTO 30
680 GOTO 320
900 REM IN GROT
910 PRINT "WE BEVINDEN ONS IN EEN ENGE GROT
WAARDOOR EEN KREEKJE STROOMT"
920 PRINT "MIDDEN IN DE GROT LIGT EEN PRACHTIG
ZWAARD"
930 PRINT "U KUNT DIT ZWAARD PAKKEN (1) DAN WEL
HET LATEN LIGGEN (2)"
935 INPUT KEUZE
940 IF KEUZE=1 THEN A$="ZWAARD"
```

```
950 PRINT "WE KUNNEN NU TERUG GAAN (1) DAN WEL IN
HET KREEKJE SPRINGEN (2)"
955 INPUT KEUZE
960 IF KEUZE=2 THEN PRINT "HET WATER IS VEEL TE
KOUD...":GOTO 2000
970 PRINT " WE KUNNEN NU NOG VERDER TERUGGAAN TOT
VOOR DE RIVIER (1)"
980 PRINT "DAN WEL DE RIVIER WEER VERVOLGEN (2)"
985 INPUT KEUZE
990 IF KEUZE=1 THEN GOTO 110
1010 PRINT "PLOTSELING VERSCHIJNT EEN BOZE
TOVENAAR..."
1020 PRINT "VAAGJES HOREN WE NOG ENKELE SPREUKEN..."
1030 PRINT "LANGZAMERHAND VERANDEREN WE IN STEEN"
1040 GOTO 2000
1200 REM OUD KASTEEL
1210 PRINT "WE ZIEN NU EEN OUD KASTEEL VOOR ONS"
1220 PRINT "WE KUNNEN NU OVER DE BRUG DE POORT
INGAAN (1)"
1230 PRINT "DAN WEL DOOR DE DROGE SLOTGRACHT
GAAN (2)"
1235 INPUT KEUZE
1240 IF KEUZE=2 THEN PRINT "HELAAS DE SLOTGRACHT
WAS NIET DROOG...":GOTO 2000
1250 PRINT "WE LOPEN NU NAAR EEN BINNENPLAATS"
1260 PRINT "PLOTSELING VERSCHIJNT ER EEN
VERSCHRIKKELIJKE DRAAK"
1270 IF A$<>"ZWAARD" THEN PRINT "WE KUNNEN ONS NIET
VERDEDIGEN...":GOTO 2000
1280 PRINT "WE VERDEDIGEN ONS MET HET ZWAARD"
1290 PRINT "WE VERSLAAN DE DRAAK MET HET ZWAARD"
1300 PRINT "ACHTER DE DRAAK ZIEN WE NU TWEE DEUREN"
1310 PRINT "WE KUNNEN NU DE LINKER (1) DAN WEL DE
RECHTER DEUR (2) INGAAN"
1315 INPUT KEUZE
1320 IF KEUZE=2 THEN PRINT "WE VALLEN IN DE
DIEPTE...":GOTO 2000
1330 PRINT "VOOR U LIGT DE SCHAT"
1340 PRINT "HET AVONTUUR IS VOOR U GELUKKIG
AFGELOPEN"
1350 PRINT "PROFICIAT"
1360 GOTO 3000
2000 PRINT "HELAAS HET AVONTUUR HEEFT VOOR U EEN
SLECHTE WENDING GENOMEN"
2010 PRINT "U KUNT HET NATUURLIJK NOG EEN KEER
PROBEREN"
2020 PRINT "DOOR HET RUN-COMMANDO TE GEVEN"
2030 GOTO 3000
3000 PRINT "EINDE SPEL"
3010 END
```

De ON...GOTO-instructie

In een aantal gevallen willen we op grond van een bepaald tussenresultaat naar een bepaald regelnummer springen. Nu zouden we een dergelijke constructie best met een aantal IF...THEN- en GOTO-instructies kunnen bereiken. Onze computer beschikt over een speciaal programmeergereedschap en wel in de vorm van de zogenaamde ON...GOTO-instructie. We behandelen deze instructie aan de hand van het volgende voorbeeld:

```
10 PRINT "1,2 OF 3";:INPUT A
20 PRINT "HET ENGELSE WOORD"
30 PRINT "VOOR DIT GETAL IS"
40 ON A GOTO 50,70,90
50 PRINT "ONE"
60 GOTO 100
70 PRINT "TWO"
80 GOTO 100
90 PRINT "THREE"
100 END
```

Voorbeeld van een resultaat:

```
1,2 OF 3?3
HET ENGELSE WOORD
VOOR DIT GETAL IS
THREE
```

De regels 10 t/m 30 spreken voor zich. Regel 40 geeft de ON...GOTO-instructie. Tussen ON en GOTO mogen we een uitdrukking plaatsen waarvan de waarde 1, 2 of 3 dient te zijn. Als de waarde 1 is, zal naar het eerste regelnummer worden gesprongen dat na GOTO staat aangegeven. Als de uitdrukking gelijk is aan 2, zal naar het tweede regelnummer worden gesprongen. In ons geval staat er geen ingewikkelde uitdrukking tussen ON en GOTO, maar direct de variabele A waarvan dus de waarde 1, 2 of 3 dient te zijn.
Het is ten slotte verleidelijk om een kleine 'zonde' te begaan. We toetsen na het RUN-commando in:

```
-3
```

Met een afkeurend gebaar toont de computer ons:

```
ERROR- 3 AT LINE 40 (VALUE ERROR)
```

Uiteraard zien we een foutmelding: we hadden immers alleen maar 1, 2 of 3 mogen intoetsen.

9. De FOR . . . NEXT-instructie

Inleiding

In de nu volgende hoofdstukken zullen we kennismaken met twee nieuwe zaken, namelijk de FOR...NEXT-instructie (hoofdstuk 9) en de array (hoofdstuk 10). De FOR...NEXT-instructie is een handige notatiewijze om aan te geven dat een bepaalde reeks instructies een aantal keren herhaald moet worden afgewerkt.
In feite is de FOR...NEXT-instructie geen wezenlijke uitbreiding van ons programmeergereedschap, omdat, zoals we ook zullen tonen, eenzelfde resultaat tevens met behulp van een IF...THEN-instructie kan worden bereikt.
Dit geldt niet voor de array: dit is wel degelijk een nieuw concept. De

array biedt ons de mogelijkheid om met een eenvoudige uitdrukking een groot aantal variabelen te introduceren. We zullen tevens zien hoe de array en FOR...NEXT-instructie bij elkaar passen.
We starten nu met de FOR...NEXT-instructie.

FOR...NEXT-instructie

Ter introductie van de FOR...NEXT-instructie bespreken we eerst een programma waarin een bepaald aantal instructies herhaald, dat wil zeggen een aantal malen, moet worden afgewerkt.
Stel bijvoorbeeld eens dat we van de serie opeenvolgende waarden 1, 2, 3, 4 en 5 het kwadraat willen afdrukken. (N.B. Geen krachttoer, maar het zal duidelijk zijn dat het ons hier om het programmeren gaat.) Het volgende programma toont een voorbeeld van een constructie waarmee dat mogelijk is. Let wel, dit programma bevat nog geen FOR...NEXT-instructie, maar het toont ons een structuur waarbij zal blijken dat FOR...NEXT-instructies bij uitstek van pas komen.

```
10 K=0
20 K=K+1
30 K2=K*K
40 PRINT K,K2
50 IF K<5 THEN 20
60 PRINT "EINDE"
70 END
```

Na het RUN-commando verschijnt:

```
1        1
2        4
3        9
4        16
5        25
EINDE
```

We zullen dit programma nu regel voor regel bespreken. Regel 10 krijgt de startwaarde 0. Regel 20 toont weer de bekende constructie:

```
K=K+1
```

Als gevolg van deze instructie wordt K met 1 verhoogd. Zo zal de eerste keer K de waarde 1 krijgen, de tweede keer de waarde 2, de derde keer 3 enz. Regel 30 introduceert een nieuwe variabele, namelijk K2 waaraan de waarde K x K, met andere woorden het kwadraat van K, wordt toegekend. In de daarop volgende instructie worden K en K2 afgedrukt. Nu komt het... we willen in dit voorbeeld de kwadraten van 1, 2, 3, 4 en 5 afdrukken, met andere woorden we willen de groep instructies van regel 30 en 40 een aantal malen herhalen. Dit kunnen we dan aangeven door in een IF-instructie te testen of we alle waarden, waarvoor het kwadraat moet worden uitgerekend, al hebben gehad. Als dit nog niet het geval is, dient de computer naar regel 20 terug te springen. Als eerste

actie zal K weer met 1 worden verhoogd enz. We zullen nu laten zien hoe hetzelfde resultaat met behulp van een FOR...NEXT-instructie kan worden bereikt. Het 'identieke' programma met een FOR...NEXT-instructie luidt:

```
10 FOR K=1 TO 5
20 K2=K*K
30 PRINT K,K2
40 NEXT K
50 PRINT "EINDE"
60 END
```

Het programma begint met de uitdrukking:

```
FOR K=1 TO 5
```

Deze eerste uitdrukking geeft aan dat alle instructies die liggen tussen deze instructie en de regel met:

```
NEXT K
```

voor de waarden K=1, K=2, K=3, K=4 en K=5, moeten worden uitgevoerd. De variabele die na de term FOR wordt aangegeven, noemen we om voor de hand liggende redenen de telvariabele. Merk op dat deze variabele kennelijk ook in het gedeelte tussen FOR en NEXT mag worden gebruikt: K komt immers in regel 20 en 30 voor. In het algemeen luidt de eerste regel bij een FOR...NEXT-instructie:

FOR naam variabele = beginwaarde TO eindwaarde

De regel die het gedeelte dat steeds moet worden herhaald afsluit, luidt:

NEXT naam variabele

De winst ten opzichte van het eerste programma zal duidelijk zijn. Bij het eerste programma moeten we eerst nagaan wat de conditie in de IF-instructie tot gevolg zal hebben en zal een eventuele sprong naar een vorige instructie volgen. Het programma met de FOR...NEXT-instructie is recht toe recht aan en als gevolg hiervan beter leesbaar.
Het volgende programma toont ons hoe we met de FOR...NEXT-instructie eenvoudig een programma kunnen construeren om het gemiddelde van een reeks getallen te bepalen.

```
10 SOM=0
20 PRINT "AANTAL=";:INPUT AANTAL
30 FOR K=1 TO AANTAL
40 PRINT "VOER GETAL ";K;" IN"
50 INPUT G
60 SOM=SOM+G
70 NEXT K
80 GEMIDDELDE=SOM/AANTAL
90 PRINT "GEMIDDELDE=";GEMIDDELDE
100 END
```

In regel 10 wordt aan SOM weer de startwaarde 0 gegeven. Eigenlijk is dat met onze computer niet nodig; de beginwaarde van een variabele is namelijk altijd 0. De getoonde programmeerwijze is echter netter. Bij deze variabele zullen we steeds alle in te voeren getallen optellen, zodat SOM uiteindelijk de som van die getallen zal bevatten.
In regel 20 wordt gevraagd, het aantal getallen waarvan het gemiddelde moet worden bepaald op te geven. Dit getal zal dan aan de variabele AANTAL worden toegekend. Hierna volgt de FOR...NEXT-instructie. De regels 40, 50 en 60 zullen nu voor de waarden K=1,2,3, ... tot AANTAL worden herhaald. Terzijde merken we hierbij op dat we hier ook wel eens van een lus spreken. Steeds volgt een melding welk getal moet worden ingevoerd (regel 40) en deze waarde wordt dan aan G toegekend (regel 50). Hierna wordt deze waarde bij SOM opgeteld (regel 60). Zodra alle waarden zijn ingevoerd, wordt in regel 80 het gemiddelde berekend en deze waarde wordt dan als resultaat van de PRINT-instructie op regel 90 afgedrukt.
Voorbeeld van een resultaat:

```
AANTAL=?4
VOER GETAL 1 IN
?3
VOER GETAL 2 IN
?5
VOER GETAL 3 IN
?4
VOER GETAL 4 IN
?4
GEMIDDELDE= 4
```

STEP

We mogen in een FOR...NEXT-instructie ook aangeven met welke waarde de telvariabele steeds weer zal worden verhoogd. Hierbij maken we dan gebruik van de term STEP. Het volgende voorbeeld verduidelijkt dit:

```
10 FOR K=2 TO 10 STEP 3
20 PRINT K
30 NEXT K
40 PRINT "EINDE LUS"
50 PRINT K
60 END
```

Resultaat:

```
2
5
8
EINDE LUS
11
```

Merk op hoe K hier achtereenvolgens de waarden 2, 5 en 8 krijgt. Tijdens

de laatste maal dat de lus wordt doorlopen krijgt K de waarde 8+3=11 en aangezien deze waarde groter is dan de aangegeven 'eindwaarde' 10, zal de lus niet opnieuw worden doorlopen. In het programma wordt dit geïllustreerd door in regel 50 nogmaals de waarde K af te drukken. We zien dat inderdaad waarde 11 wordt afgedrukt. Het zal de lezer ondertussen wel duidelijk zijn dat we bij het getal 10, in de regel die met FOR begint, beter van *grenswaarde* dan van eindwaarde kunnen spreken. Eindwaarde zou immers kunnen inhouden dat dit steeds de laatste waarde is die K zal aannemen terwijl we uit het voorbeeld kunnen opmaken dat K deze waarde binnen de lus in feite nooit aanneemt!
Het volgende voorbeeld toont ons dat we ook negatieve stappen mogen aangeven:

```
10 FOR K=10 TO 2 STEP -3
20 PRINT K
30 NEXT K
40 PRINT "EINDE LUS"
50 PRINT K
60 END
```

Resultaat:

```
10
7
4
EINDE LUS
1
```

Na de uitleg van het vorige programma zal dit programma stellig geen problemen meer veroorzaken.
Ten slotte willen we opmerken dat beginwaarde, grenswaarde en stapgrootte ook met uitdrukkingen mogen worden aangegeven. Het volgende voorbeeld illustreert dit:

```
10 A=1
20 B=2
30 C=3
40 FOR K=A TO (A+B)*C STEP (A+C)/B
50 PRINT K
60 NEXT K
70 END
```

Resultaat:

```
1
3
5
7
9
```

De lezer kan als oefening zelf nagaan of dit alles klopt.

Geneste lussen

Een FOR...NEXT-instructie kan op zich weer een andere FOR...NEXT-instructie (en deze ook weer een andere, enz.) oproepen, op voorwaarde dat er nooit een overlapping optreedt. In schema:

```
┌──── FOR K ...           ┌──── FOR K ...
│     ...                 │     ...
│  ┌─ FOR J ...        ┌──┼──── FOR J ...
│  │  ...              │  │     ...
│  └─ NEXT J           │  └──── NEXT K
│     ...              │        ...
└──── NEXT K           └─────── NEXT J
```

geen overlapping =
correcte constructie

wel overlapping =
niet toegestaan

Als een FOR...NEXT-instructie een andere FOR...NEXT-instructie bevat, spreken we van een 'geneste lus'. Het volgende programma toont een voorbeeld:

```
10 FOR K=1 TO 2
20 PRINT K
30 FOR J=1 TO 4
40 PRINT J;
50 NEXT J
60 PRINT
70 NEXT K
80 END
```

Resultaat:

```
1
1 2 3 4
2
1 2 3 4
```

Het programma start met een FOR...NEXT-instructie die zich uitstrekt van regel 10 tot 70. Binnen deze FOR...NEXT-instructie is een tweede FOR...NEXT-instructie opgenomen, die zich uitstrekt van regel 30 tot 50. We kunnen zien hoe voor iedere waarde van K steeds alle waarden van J worden afgedrukt.
Op dit moment zullen de directe toepassingen van dergelijke constructies nog wel niet zo duidelijk zijn. Na de behandeling van de array zal dat echter stellig wel het geval zijn.

Het maken van grafieken

De FOR...NEXT-instructie vormt ook een prachtig gereedschap voor het afbeelden van figuren. In het nu volgende tonen we hoe met behulp

van de PRINT-instructie een curve op het scherm kan worden getekend. We starten onmiddellijk met het programma:

```
10 REM PLOT VAN SINUS
20 PRINT "ESC CTRL/CLEAR"     (zie hoofdstuk 6)
30 FOR X=0 TO 21
40 Y=SIN(2*3.1415*X/22)
50 REM AANPASSEN AAN SCHERM
60 REM MET BETREKKING TOT PRINT-INSTRUCTIE
70 Y=19+19*Y
80 REM WAARDE AFRONDEN
90 Y=INT(Y+0.5)
100 REM GEBRUIK Y IN TAB-FUNCTIE
110 POSITION Y,X:PRINT "*"
120 NEXT X
130 END
```

Resultaat:

```
                                   *
                                      *

                                      *
                                   *
                               *
                         *
                     *
                *
           *
      *
   *
*

*
   *
      *
          *
                *

READY
```

We zien hoe hier van de FOR...NEXT-instructie gebruik wordt gemaakt en wel om precies 22 waarden op het scherm af te drukken. Het programma is rijkelijk van commentaar voorzien zodat het niet moeilijk zal zijn een procedure te doorzien.

10. De array

We gaan wederom uit van een reeks getallen en stellen ons nu voor, om aan de hand van een programma de vraag op te lossen of er tenminste twee getallen aan elkaar gelijk zijn. Nu zou het kunnen zijn dat toevallig het eerste en het laatste getal aan elkaar gelijk zijn. Dit houdt in dat we tenminste het eerste getal tijdelijk moeten onthouden en wel tot het moment dat het laatste getal wordt ingevoerd. Anders kunnen we deze getallen immers niet met behulp van de computer vergelijken.
Hetzelfde geldt trouwens voor het tweede getal en het derde enz. De moraal van dit liedje is eenvoudig dat we alle getallen afzonderlijk moeten opslaan. Zouden we dan toch de ongelukkige constructie

```
10 INPUT GETAL1
20 INPUT GETAL2
30 INPUT GETAL3
enz.
```

moeten gebruiken? Deze constructie is stellig onhandelbaar als het gaat om een relatief groot aantal getallen. Denk bovendien eens aan al die IF...THEN-instructies om die getallen onderling te vergelijken, brrr...
BASIC biedt voor dit probleem gelukkig een zeer elegante oplossing en wel in de vorm van een zogenaamde array.
Bij een array worden er door middel van een enkele instructie een willekeurig groot aantal variabelen geïntroduceerd. De praktische beperking wordt gewoonlijk alleen door de hoeveelheid RAM-geheugen bepaald. Al die variabelen mag men dan als 'gewone' variabelen behandelen. In het nu volgende zullen we eerst in het algemeen de array behandelen aan de hand van een eenvoudig voorbeeld. Hierna wordt de oplossing van het zojuist genoemde probleem gegeven.
De instructie om de variabelen te introduceren is de DIM-instructie. Deze heeft steeds de volgende gedaante:

DIM naam van variabele (aantal)

Bijvoorbeeld:

```
DIM A(20)
```

In dit geval mag men de variabelen met de volgende namen gebruiken:

```
A(1), A(2), A(3), ... A(20)
```

Bovendien mogen we bij de Atari nog de variabele A(0) gebruiken. Al deze variabelen, kortom A(0) t/m A(20), vormen dan de array (Nederlands: reeks of rij). Deze variabelen, ook wel 'de elementen van de array' genoemd, mag men dan zoals gezegd als iedere andere variabele gebruiken. Zo zijn alle volgende uitdrukkingen toegestaan:

```
SOM=A(1)+24
A(2)=A(2)+SQR(A(4))
A(4)=A(3)^2+A(19)
```

De letter, in dit geval A, noemt men de hoofdnaam en het getal tussen haakjes noemen we de index. We zouden de hoofdnaam ook wel met een 'familienaam' kunnen vergelijken. Voor het geven van deze familienaam gelden dan weer de regels voor het geven van namen. Men mag de index ook indirect door middel van een variabele, ja zelfs door middel van een uitdrukking aangeven. Zo zijn er dus drie situaties:

A(3)	:	index wordt direct door getal aangegeven.
A(K)	:	index wordt indirect door variabele aangegeven.
A(K+4)	:	index wordt indirect door een uitdrukking aangegeven.

We kunnen ons best voorstellen dat u tot op dit moment de werkelijke waarde van de array ontgaat. Om een en ander meer tastbaar te maken, starten we met een eenvoudig voorbeeld. In dit voorbeeld tonen we dat we inderdaad een rij waarden aan een array kunnen toekennen...

```
10 DIM X(10)
20 FOR K=1 TO 10
30 INPUT A:X(K)=A
40 NEXT K
50 PRINT "WELKE WAARDE WILT U ZIEN?"
60 INPUT W
70 PRINT "DIE WAARDE WAS ";X(W)
80 PRINT "DE SOM VAN DE EERSTE"
90 PRINT "EN LAATSTE WAARDE IS"
100 SOM=X(1)+X(10)
110 PRINT SOM
120 END
```

Voorbeeld van een resultaat. Na het RUN-commando worden tien willekeurige waarden ingevoerd, bijvoorbeeld:

```
?13
?17
?21
?37
?11
?25
?23
?12
?19
?47
```

Hierna verschijnt onmiddellijk:

```
WELKE WAARDE WILT U ZIEN?
?3
DIE WAARDE WAS 21
DE SOM VAN DE EERSTE
EN LAATSTE WAARDE IS
60
```

In regel 10 wordt door middel van de DIM-instructie ruimte voor 10 variabelen gereserveerd (als we X(0) meerekenen zelfs 11 variabelen). De hoofdnaam is in dit geval X. Op deze wijze worden dus de variabelen X(1), X(2), ... X(10) en bovendien X(0) geïntroduceerd. Regel 20 geeft het begin van een FOR...NEXT-instructie aan. Als gevolg van regel 30, die in de FOR...NEXT-instructie is opgenomen, zal tienmaal een vraagteken verschijnen en wel ten teken dat er een waarde moet worden ingevoerd. Wellicht had u de instructie: INPUT X(K) verwacht. Het is bij de Atari echter verboden een array-variabele in een INPUT-instructie aan te geven. De geboden oplossing wordt daarom aanbevolen. De eerste maal (K=1) zal zo een waarde aan X(1) worden toegekend, de tweede maal een waarde aan X(2) enz. Zo worden dan aan de variabelen X(1) t/m X(10) de aangegeven waarden toegekend. Om te demonstreren dat al deze verschillende waarden ook werkelijk worden onthouden, mag u een willekeurige waarde opvragen (regel 50-70). Ten slotte wordt gedemonstreerd dat we deze variabelen als de reeds besproken variabelen mogen gebruiken en wel aan de hand van de instructie op regel 100; hier worden twee elementen van de array opgeteld. Nu dan het oorspronkelijke probleem: het herkennen of in een rij getallen tenminste twee getallen aan elkaar gelijk zijn. We gaan er van uit dat een rij getallen weer met een FOR...NEXT-instructie zal worden ingevoerd:

```
10 PRINT "VOER AANTAL IN":INPUT AANTAL
20 REM RESERVEER RUIMTE
30 DIM A(AANTAL)
40 FOR K=1 TO AANTAL
50 PRINT "VOER GETAL NR ";K;" IN"
60 INPUT X:A(K)=X
70 NEXT K
```

Dit gedeelte zal geen problemen meer opleveren; de constructie is in feite identiek aan de reeds besproken constructie. Merk op hoe in regel 30 ruimte voor de variabelen A(1), A(2) t/m A(AANTAL) wordt gereserveerd.
Hoe kunnen we nu ontdekken of er tenminste twee getallen aan elkaar gelijk zijn? We zouden kunnen beginnen met het eerste getal, met andere woorden A(1), om dit getal dan vervolgens te vergelijken met A(2), A(3) tot en met A(AANTAL). Als bij dit vergelijken twee getallen aan elkaar gelijk blijken te zijn, kunnen we een sprong maken naar een PRINT-instructie die de boodschap

```
TWEE GETALLEN ZIJN GELIJK
```

afdrukt. Als deze situatie niet optreedt, kunnen we het eerste getal wel vergeten en we herhalen de gehele procedure vervolgens met het

tweede getal. Mocht ook dit geen succes opleveren dan vergeten we tevens het tweede getal en vervolgen onze procedure met het derde getal enz.
Bovenstaande werkwijze kan men bondig met twee FOR...NEXT-instructies aangeven, namelijk:

```
80 FOR K=1 TO AANTAL-1
90 FOR J=K+1 TO AANTAL
100 IF A(K)=A(J) THEN 150
110 NEXT J
120 NEXT K
130 PRINT "ER ZIJN GEEN TWEE GETALLEN GELIJK"
140 STOP
150 PRINT "TWEE GETALLEN ZIJN GELIJK"
160 END
```

De kern van de oplossing wordt hier door de regels 80-120 gevormd. Iedere keer zal A(K) met de waarden A(K+1), A(K+2) enz. worden vergeleken. Als er een gelijkheid wordt gevonden, vindt de sprong naar de PRINT-instructie op regel 150 plaats. Als alle getallen zijn vergeleken en er treedt geen gelijkheid op, dan wordt eerst de instructie op regel 130 uitgevoerd en hierna stopt het programma bij regel 140. Hiermee is de volledige oplossing van ons probleem gegeven.

Enkele opmerkingen rond arrays

Bij het werken met arrays dienen we met enkele afspraken rekening te houden. In het nu volgende zullen we deze afspraken kort bespreken.

over namen: in een programma mag men voor een array en een 'gewone' variabele dezelfde naam gebruiken.

plaats DIM: de DIM-instructie dient steeds vooraf te gaan aan het werkelijk werken met de afzonderlijke array-variabelen. Gewoonlijk plaatst men de DIM-instructies vooraan in het programma.

weglaten van DIM: laat men de DIM-instructie weg en hanteert men een uitdrukking als A(3) dan wordt automatisch DIM A(10) verondersteld.

meer DIM's op één regel: men mag meer DIM-instructies op één regel plaatsen, bijvoorbeeld:

```
10 DIM A(30),B(300)
```

bij READ: we mogen bij een READ-instructie geen array-elementen opgeven. Zo is:

```
10 READ A(1),A(2)
```

niet toegestaan.

Zoekacties bij gekoppelde lijsten

Bij een voorraadadministratie hebben we artikelen en aantallen. Op grond van bepaalde eisen rond de aantallen moeten we al of niet artikelen bijbestellen. Dit probleem leidt tot het zogenaamde probleem van zoekacties bij gekoppelde lijsten.
In het nu volgende programmavoorbeeld gaan we er van uit dat array A de codes (getallen) van de artikelen bevat en array B de aantallen per artikel. Bijvoorbeeld als A(3)=417 en B(3)=14, dan geven deze waarden aan dat er van het artikel met artikelcode 417 nog 14 'stuks' voorradig zijn. Het volgende voorbeeld toont hoe we kunnen nagaan, van welke artikelen er minder dan tien stuks voorradig zijn:

```
10 DIM A(20),B(20)
20 FOR K=1 TO 20
30 PRINT "ARTIKELCODE=";
40 INPUT X:A(K)=X
50 PRINT "AANTAL=";
60 INPUT X:B(K)=X
70 NEXT K
80 FOR K=1 TO 20
90 IF B(K)<10 THEN PRINT A(K)
100 NEXT K
110 END
```

Merk allereerst op hoe regel 10 twee arrays introduceert. In het daarop volgende gedeelte (regel 20 t/m 50) worden alle gegevens ingevoerd. De regels 60 t/m 80 beschrijven de zoekactie; een zoekactie die neerkomt op een eenvoudige IF-instructie die in een FOR...NEXT-instructie is opgenomen. De lezer zal bij dit voorbeeld wellicht opmerken: 'Als ik die gegevens toch één voor één moest invoeren, dan kan ik toch meteen nagaan of er per artikel minder dan tien stuks aanwezig zijn...?'. Dit is juist, maar we moeten ons voorstellen dat we ook programma's kennen waarin we slechts eenmaal alle gegevens hoeven in te voeren en waarbij we die gegevens vervolgens bijvoorbeeld op floppy disk opslaan. In de praktijk zullen de regels 20 t/m 50 dan ook worden vervangen door een programma dat deze waarden van floppy disk inleest en aan de arrays A(K) en B(K) toekent.

Berekenen van minimum en maximum

Bij veel berekeningen komt de vraag voor, wat het maximum en het minimum van een gegeven reeks is. Het volgende programma toont de oplossing:

```
10 PRINT "AANTAL=";:INPUT AANTAL
20 DIM A(AANTAL)
30 FOR K=1 TO AANTAL
40 INPUT X:A(K)=X
50 NEXT K
60 REM NU VOLGEN ZOEKACTIES
```

```
70 MIN=A(1)
80 MAX=A(1)
90 FOR K=1 TO AANTAL
100 IF A(K)>MAX THEN MAX=A(K)
110 IF A(K)<MIN THEN MIN=A(K)
120 NEXT K
130 PRINT "MIN=";MIN,"MAX=";MAX
140 END
```

De regels 10 t/m 50 betreffen het invoeren van gegevens. De regels 60 t/m 120 beschrijven het gedeelte waarin de minimale en de maximale waarde worden bepaald. Uiteindelijk zal MIN de minimale en MAX de maximale waarde bevatten. De variabelen MIN en MAX krijgen eerst de startwaarde A(1). Hierna worden de opeenvolgende waarden A(K) met MIN en MAX vergeleken. Iedere keer als een nieuw minimum of maximum wordt gevonden, wordt de betreffende variabele bijgesteld (regels 100 en 110).

Sorteren

Sorteren is een van de meest voorkomende acties bij administratieve programma's. Er bestaan zeer veel sorteermethoden. Het nu volgende programma staat bekend onder de naam 'bubble sort'-methode. Hierbij worden de opeenvolgende waarden één voor één met elkaar vergeleken en als de volgende waarde groter is, vindt verwisseling plaats.
Door dit proces een aantal keren te herhalen, bubbelen als het ware alle 'lichte' getallen naar boven en zakken alle 'zware' getallen naar beneden. Het programma luidt:

```
10 PRINT "AANTAL IS";:INPUT N
20 DIM A(N)
30 FOR K=1 TO N
40 INPUT X:A(K)=X
50 NEXT K
60 REM NU VOLGT SORTEREN
70 FOR K=1 TO N-1
80 FOR J=1 TO N-K
90 V=A(J)
100 W=A(J+1)
110 IF V<=W THEN 140
120 A(J)=W
130 A(J+1)=V
140 NEXT J
150 NEXT K
160 REM NU UITVOEREN
170 PRINT "GESORTEERD"
180 FOR K=1 TO N
190 PRINT A(K)
200 NEXT k
210 END
```

Voorbeeld:

```
AANTAL IS?5
?4
?6
?3
?7
?5
GESORTEERD
3
4
5
6
7
```

Om na te kunnen gaan hoe de procedure nu precies in zijn werk gaat, kan men tussen regel 140 en 150 het volgende opnemen:

```
141 FOR M=1 TO N
142 PRINT A(M);" ";
143 NEXT M
144 PRINT
```

We raden de lezer aan dit ook echt te doen, niets is zo leerzaam als zaken tot de bodem toe uit te zoeken!

Twee-dimensionale arrays

We kunnen, na de hoofdnaam in een DIM-instructie, ook twee cijfers en wel gescheiden door een komma plaatsen. Plaatst men twee cijfers tussen de haakjes dan gaat het om een twee-dimensionale array.
In het nu volgende geven we allereerst een eenvoudige demonstratie met betrekking tot een twee-dimensionale array en vervolgens tonen we nog een spelletje. Beschouw als voorbeeld de instructie:

```
DIM A(3,4)
```

Hiermede worden de volgende variabelen geïntroduceerd:

```
A(0,0)  A(0,1)  A(0,2)  A(0,3)  A(0,4)
A(1,0)  A(1,1)  A(1,2)  A(1,3)  A(1,4)
A(2,0)  A(2,1)  A(2,2)  A(2,3)  A(2,4)
A(3,0)  A(3,1)  A(3,2)  A(3,3)  A(3,4)
```

Deze variabelen (elementen) kan men dan weer op dezelfde wijze behandelen als de variabelen bij de tot nu toe behandelde één-dimensionale array. Bijvoorbeeld:

```
10 DIM A(3,4)
20 FOR K=1 TO 3
30 FOR J=1 TO 4
```

```
40 A(K,J)=10*K+J
50 PRINT A(K,J);" ";
60 NEXT J
70 PRINT
80 NEXT K
90 END
```

Na het RUN-commando verschijnt:

```
11 12 13 14
21 22 23 24
31 32 33 34
```

Spelletje

Ter afsluiting tonen we een spelletje. Bij dit spelletje gaan we uit van een vierkant met getallen:

?	?	?
?	?	?
?	?	?

Elk vraagteken stelt een getal voor dat u moet raden en wel op grond van de volgende informatie. Eerst zien we de sommen van alle horizontale lijnen (van boven naar beneden) en vervolgens zien we de sommen van de verticale rijen (van links naar rechts). Vervolgens gaan we de getallen raden en wel in de volgende volgorde:

1 2 3
4 5 6
7 8 9

De computer zal vervolgens tonen welk getal goed geraden is. Bijvoorbeeld als u voor alle negen posities een 1 invult, ziet u op welke plaatsen werkelijk een 1 stond. Als alle getallen eenmaal geraden zijn, stopt de computer en toont u het aantal beurten dat nodig was. Programma:

```
10 DIM A(3,3), B(3,3)
20 REM BEPAAL WILLEKEURIGE GETALLEN <10
30 FOR K=1 TO 3
40 FOR J=1 TO 3
50 R=INT(RND(1)*9+1)
60 A(K,J)=R
70 NEXT J
80 NEXT K
90 PRINT "DE SOM VAN DE HORIZONTALE RIJEN IS"
100 PRINT "VAN BOVEN NAAR BENEDEN"
110 FOR K=1 TO 3
120 SOM=0
130 FOR J=1 TO 3
```

```
140 SOM=SOM+A(K,J)
150 NEXT J
160 PRINT SOM
170 NEXT K
180 PRINT "DE SOM VAN DE VERTICALE RIJEN IS"
190 PRINT "VAN LINKS NAAR RECHTS"
200 FOR J=1 TO 3
210 SOM=0
220 FOR K=1 TO 3
230 SOM=SOM+A(K,J)
240 NEXT K
250 PRINT SOM
260 NEXT J
270 BEURTEN=0
280 PRINT "NU GAAN WE RADEN"
290 PRINT "IN DE VOLGENDE VOLGORDE"
300 PRINT "1  2  3"
310 PRINT "4  5  6"
320 PRINT "7  8  9"
330 BEURTEN=BEURTEN+1
340 FOR K=1 TO 3
350 FOR J=1 TO 3
360 PRINT "RAAD GETAL ";J+(K-1)*3
370 INPUT X:B(K,J)=X
380 NEXT J
390 NEXT K
400 REM WAT IS GOED?
410 TEL=0
420 FOR K=1 TO 3
430 FOR J=1 TO 3
440 IF A(K,J)<>B(K,J) THEN 480
450 PRINT A(K,J);" ";
460 TEL=TEL+1
470 GOTO 490
480 PRINT " ";"*";
490 NEXT J
500 PRINT
510 NEXT K
520 IF TEL=9 THEN 540
530 GOTO 280
540 PRINT "GERADEN IN";BEURTEN;" BEURTEN"
550 END
```

11. Subroutines: GOSUB en RETURN

GOSUB

Om het belang van de GOSUB-instructie uit te leggen, kijken we eerst naar het schaakspel. Het schaakspel is stellig een van de meest uitdagende spellen voor bepaalde programmeurs. Ieder jaar worden er toernooien 'computerschaak' gehouden. Het aantal schaakcomputers, of beter, het aantal schaakprogramma's groeit met de dag.
Bij het schaken wensen we na iedere zet het bord weer te geven. Het is dan plezierig, dat we dit afbeelden van het bord in een afzonderlijk programma kunnen beschrijven, waarbij we dan iedere keer als het bord moet worden weergegeven, van ons 'hoofdprogramma' naar dit 'deelprogramma' springen. Als het deelprogramma is afgewerkt moet weer naar het hoofdprogramma worden gesprongen, alsof er niets gebeurd is.
Gewoonlijk noemen we een afzonderlijk programmagedeelte, waar tijdelijk naar toe wordt gesprongen, een subroutine. Het programma van waaruit dit gedeelte wordt opgeroepen noemen we inderdaad het hoofdprogramma.

Een subroutine is een programmagedeelte dat vanuit een willekeurige regel in het hoofdprogramma kan worden opgeroepen.

BASIC biedt voor een dergelijke constructie de GOSUB-instructie. Deze heeft steeds de volgende vorm:

```
GOSUB regelnummer
```

Het effect van deze instructie is dat naar het aangegeven regelnummer wordt gesprongen, waar het programma wordt voortgezet. Als zodanig is er geen verschil met de GOTO-instructie.
Dat verschil is er echter wel degelijk, omdat de computer, eenmaal aangekomen bij de instructie:

```
RETURN
```

automatisch weer terugkeert naar het hoofdprogramma en wel naar de regel die volgt op de regel waar de GOSUB-instructie stond.
In de afbeelding maken we de procedure, aan de hand van een schema,

duidelijk:

```
10 ...                                ┐
20 ...                                │
   ...                                │
   ...                                │
100 GOSUB 300                         │
110 ...                               │
    ...                               ├── hoofdprogramma
    ...                               │
200 GOSUB 300                         │
210 ...                               │
    ...                               │
    ...                               │
270 STOP                              ┘
300 REM BEGIN SUBROUTINE              ┐
    ...                               ├── subroutine
    ...                               │
350 RETURN                            ┘
```

Het gebruik van subroutines.

Het hoofdprogramma beslaat de regels 10 t/m 270. Op regel 100 wordt voor de eerste maal de subroutine aangeroepen. De computer vervolgt het programma met regel 300 enz. Eenmaal bij regel 350 aangekomen (RETURN), keert de computer weer terug naar het hoofdprogramma en wel naar de regel die volgt op de GOSUB-instructie, waarmee de subroutine werd opgeroepen. Zo keert de computer uiteindelijk terug bij regel 110. Bij regel 200 aangekomen, wordt weer naar de subroutine, die op regel 300 begint, gesprongen. Na het bereiken van regel 350 volgt weer een sprong terug, maar nu naar regel 210.
Merk de STOP-instructie bij regel 270 op. Op deze wijze wordt vermeden dat de computer ongewild weer met de instructies van de subroutine zou beginnen. Dit zou trouwens bij regel 350 tot een foutmelding leiden (er wordt dan namelijk een RETURN aangetroffen zonder corresponderende GOSUB-instructie).

Voorbeeld

In dit voorbeeld zullen we het berekenen van het aantal variaties tonen. We leggen eerst het begrip variaties uit. Stel dat we n verschillende voorwerpen hebben (n is bijvoorbeeld 8), dan kunnen we ons de vraag stellen op hoeveel verschillende manieren we hier rijtjes van m voorwerpen (m is bijvoorbeeld 4) uit kunnen samenstellen. We gaan er hierbij van uit, dat tevens met de volgorde rekening wordt gehouden.
Zo is de reeks a, b, c een andere reeks dan c, a, b. Welnu, het aantal mogelijke rijtjes met lengte m, uit een serie van n voorwerpen, noemen we het aantal variaties (van m uit n).
Dit aantal wordt berekend, door de volgende formule toe te passen:

$$\frac{n!}{(n-m)!}$$

Met de uitdrukking n! geven we de volgende vermenigvuldiging aan:

n x (n-1) x (n-2) x ... x 2 x 1

We duiden dit ook wel eens aan met 'de faculteit van n'. Voorbeeld: 3! = 3 x 2 x 1 = 6. Evenzo wordt met (n-m)! de volgende vermenigvuldiging aangegeven:

(n-m) x (n-m-1) x (n-m-2) x ... x 2 x 1

We zien, dat er bij onze formule tweemaal zo'n faculteit moet worden uitgerekend, namelijk de faculteit van n en de faculteit van n-m. In dit geval lijkt het zinvol om het berekenen van een faculteit in een afzonderlijke subroutine onder te brengen. De volgende routine berekent de faculteit:

```
10 PRINT "GETAL=";:INPUT F
20 GOSUB 400
30 PRINT FAC
40 STOP
400 IF F=0 OR F=1 THEN FAC=1:GOTO 450
410 FAC=F
420 FAC=FAC*(F-1)
430 F=F-1
440 IF F>1 THEN GOTO 420
450 RETURN
460 END
```

De routine wordt door de regels 400 - 450 beschreven. De eerste regels zijn enkel om een getal in te voeren, de routine aan te roepen en het resultaat af te drukken. Dit eerste programma dient dus in feite om de subroutine uit te testen.
We zien hoe de faculteit van F wordt berekend. Eerst wordt aan F een waarde toegekend (regel 10) en de subroutine wordt aangeroepen (regel 20). Op regel 400 volgt een test of F gelijk is aan 1 of 0, en indien dit het geval is, krijgt FAC de waarde 1. Tevens wordt de subroutine dan weer verlaten door een sprong naar regel 450. Terzijde merken we op dat de faculteit van 0 per definitie 1 is, vandaar dat in regel 400 tevens met 0 rekening wordt gehouden. Als F groter is dan 1 wordt FAC eerst gelijk gesteld aan F en vervolgens met F-1 vermenigvuldigd (regel 420). Hierna wordt F met 1 verlaagd (regel 430). Zolang F nog groter is dan 1, wordt het vermenigvuldigen herhaald. Probeer voor u zelf maar eens na te gaan, dat op deze manier inderdaad de faculteit van het ingevoerde getal wordt berekend.
We testen het programma uit. Na het RUN-commando toetsen we 4 in:

```
GETAL=?4
24
```

Kortom, de computer geeft als antwoord 24 en dit is inderdaad het juiste antwoord: 4 x 3 x 2 x 1 = 24. Nu we eenmaal deze routine hebben, is de rest een peuleschil.

Het volledige programma ter bepaling van het aantal variaties wordt:

```
10 PRINT "VARIATIES"
20 REM N IS AANTAL VOORWERPEN
30 REM WE VORMEN RIJTJES TER LENGTE M
40 PRINT "N=";:INPUT N
50 F=N
60 GOSUB 400
70 TELLER=FAC
80 PRINT "M=";:INPUT M
90 F=N-M
100 GOSUB 400
110 NOEMER=FAC
120 VAR=TELLER/NOEMER
130 PRINT "VARIATIES=";VAR
140 STOP

400 IF F=0 OR F=1 THEN FAC=1:GOTO 440
410 FAC=F
420 FAC=FAC*(F-1)
430 F=F-1
440 IF F>1 THEN GOTO 420
450 RETURN
460 END
```

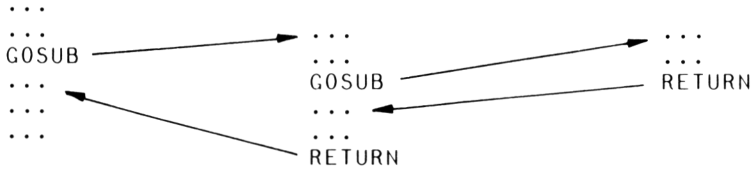
Voorbeeld:

```
VARIATIES
N=?8
M=?4
VARIATIES= 1680
```

De subroutine is exact overgenomen. De regels 40 - 70 betreffen het bepalen van de faculteit van n en evenzo bepalen regels 80 - 110 de faculteit van n-m. Hierna wordt het aantal permutaties berekend en afgedrukt.

Geneste subroutines

We merken op, dat een subroutine weer naar een volgende mag springen. Na aankomst bij RETURN zal dan worden teruggesprongen naar de subroutine van waaruit de tweede subroutine werd aangeroepen. Komen we nogmaals RETURN tegen dan keren we weer naar het hoofdprogramma terug. Als we de deelprogramma's (subroutines) naast elkaar plaatsen, wordt het volgende schema gevonden:



Op deze wijze kunnen we veel subroutines aan elkaar rijgen.

ON...GOSUB-instructie

Een afzonderlijke mogelijkheid wordt geboden door de ON...GOSUB-instructie die, wat constructie betreft, geheel en al overeenkomt met de reeds besproken ON...GOTO-instructie. Op grond van de waarde tussen ON en GOSUB (moet 1, 2, 3 enz. zijn) zal naar de eerste regel, tweede regel, derde regel enz. uit de regelnummerlijst worden gesprongen. Het volgende voorbeeld toont een wat merkwaardige illustratie:

```
10 PRINT "DE PROZA-GENERATOR"
20 PRINT
30 FOR K=1 TO 4
40 R=INT(RND(1)*5+1)
50 ON R GOSUB 80,100,120,140,160
60 NEXT K
70 STOP
80 PRINT "WEET JE WEL... IK HOU VAN JOU"
90 RETURN
100 PRINT "DE VOGELS KOMEN OOK WEER GAUW"
110 RETURN
120 PRINT "HET WATER VAN DE BEEK IS LAUW"
130 RETURN
140 PRINT "KOM BIJ ME EN TOON NOOIT BEROUW"
150 RETURN
160 PRINT "IK BLIJF JE STELLIG EEUWIG TROUW"
170 RETURN
180 PRINT
190 END
```

Voorbeeld van een resultaat:

```
DE PROZA-GENERATOR

HET WATER VAN DE BEEK IS LAUW
KOM BIJ ME EN TOON NOOIT BEROUW
DE VOGELS KOMEN OOK WEER GAUW
IK BLIJF JE STELLIG EEUWIG TROUW
```

Het 'hart' van deze proza-generator vormt regel 50. Als resultaat van de door het toeval bepaalde waarde van R zal naar een van de regels met tekst worden gesprongen. De constructie van het programma spreekt verder geheel voor zich.

12. Manipulaties met strings

In hoofdstuk 6 zagen we dat BASIC ook de mogelijkheid biedt om reeksen schrifttekens, kortom strings, aan variabelen toe te kennen. Alvorens we een string-variabele kunnen gebruiken, moeten we door middel van een DIM-instructie aan de computer bekend maken hoe groot de string, die de variabele bevat, kan worden. Als we toch een string aan een variabele toekennen, die langer is dan de opgegeven waarde bij de DIM-instructie, dan worden de schrifttekens die teveel zijn weggelaten. We illustreren dit met een voorbeeld:

```
10 DIM KORT$(5),LANG$(11)
20 KORT$="ATARI 600XL"
30 LANG$="ATARI 600XL"
40 PRINT KORT$
50 PRINT LANG$
60 END
```

Het resultaat zal zijn:

```
ATARI
ATARI 600 XL
```

De moraal is duidelijk, zorg er voor dat bij de DIM-instructie altijd een voldoende grote lengte wordt opgegeven.
Nu wordt het tijd om eens na te gaan welke bewerkingen zoal met strings

mogelijk zijn. We zullen dat doen door een overzicht van alle functies te geven. We zullen daarbij tevens zien dat er een aantal functies zijn die als resultaat weer een string opleveren. De namen van deze functies eindigen steeds op een dollarteken.
Dan is er een tweede categorie waarvan het resultaat, hoe vreemd het nu ook klinkt, een getal is. De namen van de functies van deze tweede categorie eindigen niet op een dollarteken.

String-functies

Een string-functie is een functie waarbij de 'berekende waarde' een string is, dan wel een functie waarbij tussen de haakjes na de functieterm een string wordt aangegeven.
String-functies vormen ongetwijfeld één van de grote troefkaarten in BASIC! In het nu volgende geven we een opsomming van de verschillende mogelijkheden.

LEN
Met de LEN-functie wordt het aantal schrifttekens in een string bepaald. Bijvoorbeeld:

```
LEN("MOOI")
```

levert het getal 4; de string MOOI telt immers vier tekens. Een voorbeeldprogramma:

```
10 DIM A$(100)
20 PRINT "VOER EEN NAAM IN"
30 INPUT A$
40 L=LEN(A$)
50 PRINT "DEZE NAAM IS ";
60 PRINT L;" SCHRIFTTEKENS LANG"
70 END
```

In regel 30 zien we een INPUT-instructie: nu moet echter een string worden ingevoerd. Na het commando RUN wordt eerst de tekst VOER EEN NAAM IN afgedrukt en vervolgens zien we een vraagteken om ons er aan te herinneren dat nu een string moet worden ingevoerd.
We testen het programma uit door een string in te toetsen, bijvoorbeeld:

```
TOM POES
```

Hierna toont de computer:

```
DEZE NAAM IS 8 SCHRIFTTEKENS LANG
```

Misschien had u wel de waarde 7 verwacht: de computer telt echter ook de spatie tussen TOM en POES!

ASC

Met behulp van deze functie kunnen we de eerste letter van een string in een getal omzetten. Volgens een afspraak behoort bij iedere letter, of meer in het algemeen bij ieder schriftteken, een code.
Deze codes noemen we ASCII-codes. Appendix 1 toont een overzicht van deze codes. Zo komt bijvoorbeeld de letter A met de code 65 overeen. We geven een voorbeeld:

```
10 DIM X$(10)
20 X$="AAP"
30 K=ASC(X$)
40 PRINT K
50 END
```

Na het RUN-commando verschijnt de waarde 65. Dit is inderdaad de ASCII-code die behoort bij de eerste letter (A) van de aangegeven string. De ASC-functie kunnen we bijvoorbeeld gebruiken om woorden te alfabetiseren. Een ander voorbeeld zien we bij het programma 'Geheimschrift' achter in dit boek.

CHR$

We zouden dit de 'omgekeerde' ASC-functie kunnen noemen. Het argument is nu de ASCII-code en de 'berekende waarde' het overeenkomstige schriftteken. We geven een voorbeeld:

```
10 DIM X$(1)
20 K=65
30 X$=CHR$(K)
40 PRINT X$
50 END
```

Na het RUN-commando verschijnt:

```
A
```

Dit resultaat zal ons na het voorafgaande programma niet meer zo verbazen.

STR$

Met behulp van deze functie kunnen we een getal in een string omzetten waar we dan verder allerlei string-manipulaties op kunnen toepassen. Een voorbeeld:

```
10 DIM A$(10),X$(10)
20 K=10
30 X$=STR$(K)
40 A$=X$(2)
50 PRINT A$
60 END
```

Dit programma geeft als resultaat:

```
0
```

VAL
De VAL-functie is de omgekeerde STR$-functie. Met behulp van deze functie kunnen we namelijk een string die ook als getal te lezen is, in het overeenkomstige getal omzetten. Voorbeeld:

```
10 DIM A$(10)
20 A$="22"
30 K=VAL(A$)
40 N=K*3
50 PRINT N
60 END
```

Dit programma geeft als resultaat:

```
66
```

Slicing
Slice is het Engelse woord voor plakje. We kunnen door slicing als het ware een plakje uit een string snijden. Dit plakje noemen we dan de substring. We doen dit door achter de string-variabele tussen haakjes het gedeelte aan te geven waar het om gaat. Dat wil zeggen, het nummer van het eerste en het laatste teken van de gewenste substring, gescheiden door een komma. Kijk maar eens naar het volgende voorbeeld:

```
10 DIM X$(10),A$(10)
20 X$="ATARI 600XL"
30 A$=X$(7,9)
40 PRINT A$
50 END
```

We snijden hier (regel 30) het deel '600' uit de string 'ATARI 600XL'. De uitkomst van dit voorbeeld is dus:

```
600
```

Zo kunnen we met de 'slice' (1,X) de eerste X tekens van een string verkrijgen. Als we slechts één getal tussen de haakjes zetten bedoelen we het laatste stuk van de string vanaf het aangegeven teken. Vervangen we regel 30 in bovenstaand voorbeeld door:

```
30 A$=X$(7)
```

dan zal het resultaat zijn:

```
600XL
```

Hetzelfde resultaat wordt verkregen als we voor deze regel het volgende noteren:

```
30 A$=X$(7,LEN(X$))
```

Op de volgende pagina geven we nog een voorbeeldje met slicing.

```
10 DIM NAAM$(100)
20 PRINT "Hallo, hoe heet je"
30 INPUT NAAM$
40 FOR TEKEN=1 TO LEN(NAAM$)
50 PRINT NAAM$(1,TEKEN)
60 NEXT TEKEN
70 END
```

en een voorbeeld van een resultaat:

```
Hallo, hoe heet je?        (nu wordt ALBERT ingetypt)

A
AL
ALB
ALBE
ALBER
ALBERT
```

Het geheim schuilt in de uitdrukking PRINT NAAM$(1,TEKEN) die in de FOR...NEXT-lus is opgenomen.

We kunnen strings niet zonder meer optellen. Een opdracht als A$=B$+C$ zal dus een foutmelding opleveren. Om toch strings aan elkaar te kunnen rijgen moeten we een truc toepassen. Dit gaat als volgt:

```
10 DIM A$(10),B$(10)
20 A$="COM"
30 B$="PUTER"
40 A$(LEN(A$)+1)=B$
50 PRINT A$
60 END
```

Het resultaat is:

```
COMPUTER
```

In regel 40 wordt door een slice het laatste nog ongebruikte deel van A$ aangegeven. Hierin wordt dan de string B$ geplaatst.
We zien hier hoe een substring door een andere string wordt vervangen. We moeten er dan wel op letten dat, als we slechts één getal tussen de haakjes zetten, het gedeelte van de string wordt bedoeld vanaf het aangegeven teken tot en met het einde van de string. We illustreren dit aan de hand van het volgende voorbeeld:

```
10 DIM X$(11)
20 X$="ATARI 600XL"
30 X$(6,6)="-"
40 PRINT X$
50 X$(6)="!"
60 PRINT X$
70 END
```

Het resultaat is:

```
ATARI-600XL
ATARI!
```

De uitdrukking in regel 50 heeft tot gevolg dat achter het uitroepteken de string is afgelopen.
In een aantal gevallen komt het voor dat we een string willen gebruiken die uit dezelfde tekens is opgebouwd. We geven een voorbeeld van de constructie die dan kan worden gebruikt:

```
10 DIM X$(100)
20 FOR I=1 TO 100
30 X$(I,I)="-"
40 NEXT I
50 PRINT X$
60 END
```

Veel sneller gaat dit echter met het volgende trucje:

```
10 DIM X$(100)
20 X$="-"
30 X$(100)="-"
40 X$(2)=X$
50 PRINT X$
60 END
```

Het resultaat is in beide gevallen 100 streepjes. In regel 20 wordt aan het eerste teken van X$ een streepje toegekend en in regel 30 gebeurt hetzelfde met het laatste teken van X$. De string heeft nu de lengte 100, maar we weten niet wat er tussen het eerste en het laatste teken staat. We moeten deze onbekende tekens vervangen door streepjes. Dit gebeurt in regel 40: hier wordt de string X$ in X$ zelf gezet waarbij we beginnen bij het tweede teken. Welnu, aan het tweede teken van X$ wordt door de computer nu het eerste teken van X$ toegekend. Nu zijn de eerste twee tekens beide een streepje. Dan wordt aan het derde teken het tweede teken toegekend, dat inmiddels al een streepje was geworden, waarna het vierde teken aan de beurt is en ga zo maar door. Uiteindelijk zal de hele string vol streepjes zitten.
Veel andere computers kennen geen slicing, maar gebruiken de stringfuncties LEFT$, MID$ en RIGHT$. Met de 'slicing-techniek' kunnen we deze functies gemakkelijk nabootsen, wat nog wel eens nodig kan zijn voor het omzetten van programma's van zo'n andere computer naar de Atari-computer.

LEFT$(X$,N)	wordt dan	X$(1,N)
MID$(X$,M,N)	wordt dan	X$(N,M+N-1)
RIGHT$(X$,N)	wordt dan	X$(LEN(X$)-N)

String-arrays
Helaas kunnen we niet zonder meer arrays van strings maken zoals we

dat bij getallen doen. We kunnen wel meerdere strings in één string achter elkaar plakken. We moeten dan op een of andere manier weten waar die strings in de grote string staan. We kunnen dit doen door die strings van gelijke lengte te houden, hetgeen bijvoorbeeld gebeurt in het sorteerprogrammaatje in de volgende paragraaf en in het spelletje aan het einde van dit hoofdstuk. We kunnen ook in een getallen-array bijhouden waar de strings beginnen of eindigen.

Relatie-symbolen

Hoewel we geen symbolen als +, -, * en / bij strings mogen gebruiken, kunnen we wel de tekens <,<=, >,>= en <> bij strings gebruiken. Op zich lijkt een uitdrukking als A$<B$ onzinnig, maar als we bedenken, dat aan alle tekens ook ASCII-codes (dus getallen) zijn verbonden, komt één en ander minder onwaarschijnlijk over.
Bij een uitdrukking als A$<B$ worden A$ en B$ op grond van hun alfabetische rangschikking met elkaar vergeleken. De uitdrukking A$<B$ wil dan zeggen 'A$ komt vóór B$'. Met deze wetenschap in het achterhoofd kunnen we ons sorteerprogramma van hoofdstuk 10 eenvoudig aanpassen, zodat we ook namen in alfabetische volgorde kunnen sorteren. Probeer het volgende programma maar eens:

```
10 REM SORTEERPROGRAMMA
20 PRINT "AANTAL WOORDEN"
30 INPUT AANTALW
40 DIM WRD$(10),WRDEN$(10*AANTALW),HULP$(10)
50 FOR WOORDNR=1 TO 10*AANTALW STEP 10
60 INPUT WRD$
70 WRD$(LEN(WRD$)+1)="          ":REM 10 SPATIES
80 WRDEN$(WOORDNR)=WRD$
90 NEXT WOORDNR
99 REM NU ALFABETISEREN
100 FOR K=1 TO AANTALW-1
110 FOR J=1 TO AANTALW-K
120 P1=10*(J-1)+1:P2=P1+9
130 P3=P1+10:P4=P3+9
140 IF WRDEN$(P1,P2)<=WRDEN$(P3,P4) THEN 180
150 HULP$=WRDEN$(P3,P4)
160 WRDEN$(P3,P4)=WRDEN$(P1,P2)
170 WRDEN$(P1,P2)=HULP$
180 NEXT J
190 NEXT K
199 REM NU UITVOEREN
200 PRINT "GEALFABETISEERD"
210 FOR K=1 TO 10*AANTALW STEP 10
220 PRINT WRDEN$(K,K+9)
230 NEXT K
240 END
```

Voorbeeld van een resultaat:

```
AANTAL WOORDEN
```

```
?4
? WEKKER
? AAL
? AAK
? KIKKER
GEALFABETISEERD
AAK
AAL
KIKKER
WEKKER
```

Merk op, dat de structuur van het programma inderdaad exact met het sorteerprogramma van hoofdstuk 11 overeenkomt.

Spelletje

Aan het slot van dit hoofdstuk willen we een minder serieuze, maar daarom niet minder leuke toepassing zien.
Het volgende programma toont namelijk een spelletje. Het is de bedoeling dat u een woord moet raden dat uit precies zeven letters bestaat. Enfin, toets het programma maar eens in, de aanwijzingen volgen vanzelf op het scherm.
We geven deze keer geen uitleg van het programma. Kortom, voor een regenachtige middag heeft u nu tevens een aardige puzzel.
Programma:

```
10 REM WOORDSPELLETJE
20 DIM A$(49),C$(7),D$(7),L$(7),H$(50)
30 READ A$
40 K=INT(RND(0)*7)
50 NB=0:H$=""
60 FOR N=7 TO 1 STEP -1
70 C$(8-N,8-N)=A$(7*K+N)
80 NEXT N
90 D$="*******"
100 PRINT:PRINT "HET WOORD BESTAAT UIT ZEVEN LETTERS"
110 PRINT D$
120 PRINT:PRINT "KIES EEN LETTER : ";
130 INPUT L$
140 NB=NB+1
150 H$(LEN(H$)+1)=L$
160 FOR N=1 TO 7
170 IF L$=C$(N,N) THEN D$(N,N)=L$
180 NEXT N
190 PRINT:PRINT D$
200 IF D$=C$ THEN 240
210 PRINT "AANTAL BEURTEN : ";NB
220 PRINT "GEBRUIKTE LETTERS : ";H$
230 GOTO 120
240 PRINT:PRINT "G E R A D E N !!! IN ";NB;"BEURTEN"
250 PRINT "NOG EEN KEER?";
```

```
260 INPUT L$:IF L$="J" THEN 40
270 DATA GNINEPORETSMAHLEEWULFNEOZIESNETHCAWGNIL
AWDGITSGNA
289 END
```

13. Geluid op de Atari-computer

De Atari-computer is uitgerust met een geluids-chip met vier geluidsgeneratoren. We kunnen met elke generator een apart geluid maken, zodat we vier geluiden tegelijkertijd kunnen genereren. Het geluid wordt voortgebracht door de luidspreker van de aangesloten TV. Met de instructie:

```
SOUND 0,60,10,15
```

kunnen we de hoge C laten horen en met:

```
SOUND 0,0,0,0
```

kunnen we het geluid weer uitzetten.
We zien dat bij de SOUND-instructie vier getallen moeten worden meegegeven. Deze zijn achtereenvolgens om aan te geven: welke geluidsgenerator gebruikt zal worden, de toonhoogte, de vervormingsfactor en de geluidssterkte. Algemeen ziet de SOUND-instructie er dus als volgt uit:

SOUND generatornummer,toonhoogte,vervormingsfactor,geluidssterkte

Voor 'generatornummer' kunnen we de nummers 0, 1, 2 of 3 kiezen. De toonhoogte ligt tussen 0 en 255. Wanneer we de 0 gebruiken krijgen we de hoogste toon die de computer kan voortbrengen. Bij het getal 255 behoort de laagste toon.
De vervormingsfactor is instelbaar van 0 tot 14, maar alleen de even getallen mogen worden gebruikt. De vervormingsfactor 10 zorgt voor de meest zuivere toon.
Ten slotte is de geluidssterkte regelbaar van 0 tot 15. De 0 komt overeen met 'volumeknop uit' en 15 met 'zo hard mogelijk'.
Als we meer generatoren tegelijk gebruiken moeten we er voor zorgen dat de totale geluidssterkte, dus de som van de geluidssterktes van alle in gebruik zijnde generatoren, niet boven de 32 uitkomt. Gebeurt dat wel dan zal er vervorming optreden.

Tonen

Met de geluidsgenerator kunnen we allerlei wijsjes maken. We kunnen zelfs akkoorden laten klinken. Een akkoord bestaat uit immers uit meer

tonen. Typ maar eens in:

```
10 SOUND 0,255,10,8
20 SOUND 1,173,10,8
30 SOUND 2,144,10,8
40 SOUND 3,108,10,8
50 FOR W=1 TO 700:NEXT W
60 FOR GEN=0 TO 3:SOUND GEN,0,0,0:NEXT GEN
70 END
```

Na het RUN-commando is dan gedurende een korte tijd een akkoord hoorbaar. In regel 10 tot en met 40 wordt het akkoord bepaald en wel door elk van de vier geluidsgeneratoren een toon voort te laten brengen. We gebruiken hier vervormingsfactor 10, voor het produceren van onvervormde tonen. Om de totale geluidssterkte niet boven de 32 te laten komen gebruiken we hier geluidssterkte 8. Regel 50 bevat een wachtlus waardoor het akkoord een tijdje blijft klinken. In regel 60 zetten we de generator weer uit.
De END-instructie of het einde van het programma zet trouwens ook alle toongeneratoren uit, dus strikt genomen zijn regel 60 en 70 niet nodig. Voor de duidelijkheid zullen we ze er toch steeds bij zetten. Wanneer een BASIC-programma echter op een andere manier stopt, bijvoorbeeld door de STOP-instructie of door het optreden van een fout in het programma, dan zullen de tonen die op dat moment 'aan staan', ook blijven klinken.
We kunnen met de Atari-toongenerator drie octaven bestrijken. Hieronder staat een tabel met alle tonen van deze drie oktaven, met het bij iedere toon behorende toonhoogte-getal dat in de SOUND-instructie nodig is om de betreffende toon te genereren. Verder staat bij elke toon een toets vermeld.

Tabel met tonen

noot-octaaf	toonhoogte-getal	toets Atari-orgel
C-4	29	[b]
B-3	31	b
Ais-3	33	[a]
A-3	35	a
Gis-3	37	[g]
G-3	40	g
Fis-3	42	[f]
F-3	45	f
E-3	47	e
Dis-3	50	[d]
D-3	53	d
Cis-3	57	[c]
C-3	60	c
B-2	64	B
Ais-2	68	[A]
A-2	72	A
Gis-2	76	[G]
G-2	81	G

Tabel met tonen (vervolg)

noot-octaaf	toonhoogte-getal	toets Atari-orgel
Fis-2	85	[F]
F-2	91	F
E-2	96	E
Dis-2	102	[D]
D-2	108	D
Cis-2	114	[C]
C-2	121	C
B-1	128	^B
Ais-1	136	[^A]
A-1	144	^A
Gis-1	153	[^G]
G-1	162	^G
Fis-1	173	[^F]
F-1	182	^F
E-1	193	^E
Dis-1	204	[^D]
D-1	217	^D
Cis-1	230	[^C]
C-1	243	^C

Wanneer u het volgende programma intoetst en 'runt', komen de aangegeven toetsen die we kunnen intypen overeen met de tonen. Dit programma maakt van uw Atari-computer zowaar een orgeltje.

```
10 REM HET ATARI-ORGEL
20 REM INITIALISEER TONEN
30 DIM TOON$(256)
40 TOON$=CHR$(0)
50 TOON$(256)=TOON$
60 TOON$(2)=TOON$
70 FOR I=1 TO 21
80 READ J,X,Y
90 TOON$(J,J)=CHR$(X)
100 J=J+128:TOON$(J,J)=CHR$(Y)
110 NEXT I
120 DATA 3,243,230,4,217,204,5
130 DATA 193,0,6,182,173,7,162
140 DATA 153,1,144,136,2,128,0
150 DATA 67,121,114,68,108,102,69
160 DATA 96,0,70,91,85,71,81
170 DATA 76,65,72,68,66,64,0
180 DATA 99,60,57,100,53,50,101
190 DATA 47,0,102,45,42,103,40
200 DATA 37,97,35,33,98,31,29
299 REM MAAK INGETOETSTE TONEN
300 PRINT "ESC CONTROL/CLEAR"
310 PRINT " HET ATARI-ORGEL ";
320 OPEN#2,4,0,"K:"
```

```
330 GET#2,X
340 SOUND 0,ASC(TOON$(X)),10,15
350 GOTO 330
```

De bedoeling van regel 310 is dat de tekst tussen aanhalingstekens (de te printen tekst) invers zal worden afgedrukt. Dat betekent dat de tekst in plaats van wit op zwart nu als zwart op wit moet worden afgedrukt (zie hoofdstuk 5).
Na het RUN-commando wordt 'HET ATARI-ORGEL' in inverse video zichtbaar en we kunnen dan het orgel bespelen. Bij het intoetsen van de letters A tot en met G worden de overeenkomstige tonen gespeeld. Als we in de tabel kijken dan zien we dat we kleine letters moeten intypen voor de hoge tonen uit octaaf 3 en hoofdletters voor de tonen uit octaaf 2. De lage tonen uit octaaf 1 krijgen we door de letters samen met de CONTROL-toets in te typen. Dan zijn er nog de noten Cis, Dis, Fis en Ais. Deze noten krijgen we door steeds de letter in inverse video in te typen. Als we dus op de inverse-toets drukken en vervolgens op de CONTROL-toets en tegelijkertijd op de C, dan krijgen we de Cis uit octaaf 1. We kunnen het orgel stoppen door een niet bestaande toon in te typen, bijvoorbeeld een spatie. Veel plezier bij het componeren!
Nu iets over de werking van dit programma. Eerst wordt een string, genaamd TOON$ (die we als array gebruiken), gevuld met de toonhoogtegetallen die behoren bij de toetsen. Nu worden feitelijk niet de toonhoogtegetallen opgeslagen, maar de ASCII-tekens die daar bij horen, met andere woorden CHR$(toonhoogte-getal). De plaats in de string waar de tekens worden opgeslagen wordt bepaald door de ASCII-waarde van de te gebruiken toetsen, dus ^A-^G, A-G en a-g en deze tekens in inverse video. De ASCII-waarde van een teken in inverse video is 128 groter dan de ASCII-waarde van dat teken zelf.
Nu willen we direct bij het intypen van een toets een toon horen en niet na het intypen van de RETURN-toets. We kunnen dus niet de INPUT-instructie gebruiken. We gebruiken daarom een andere constructie. Als we met de OPEN#-instructie hebben aangegeven dat we tekens van het toetsenbord, genaamd 'K:' in willen lezen, dan kunnen we met de GET#-instructie direct inlezen wat er op het toetsenbord wordt ingetypt. De GET#-instructie van regel 330 levert de ASCII-waarde van het teken op dat is ingetypt en kent dat toe aan de variabele X. In regel 340 wordt dan de toon, waarvan de hoogte door de string wordt bepaald, ten gehore gebracht.
We kunnen nu eens kijken hoe we een toon in toonhoogte kunnen laten veranderen. Probeer het volgende programmaatje maar eens uit:

```
10 FOR TOON=255 TO 1 STEP -1
20 SOUND 0,TOON,10,15
30 FOR W=1 TO 5:NEXT W
40 NEXT TOON
50 SOUND 0,0,0,0
60 END
```

We horen nu een toon die steeds hoger wordt. Dit kan een leuk effect hebben in een programma (bijvoorbeeld om aan te geven dat er iets gebeurt). Een ander bruikbaar effect is de trillende toon zoals die door het

volgende programmaatje wordt gegenereerd:

```
10 FOR I=1 TO 100
20 SOUND 0,100,10,15
30 FOR W=1 TO 20:NEXT W
40 SOUND 0,80,10,15
50 NEXT I
60 SOUND 0,0,0,0
70 END
```

Vervorming

Het is moeilijk om precies met woorden aan te geven wat het effect is van de vervormingsfactor op het geluid. Het beste is het daarom om het effect te beluisteren. Daarvoor kunnen we het volgende programma gebruiken:

```
10 PRINT " "
20 POSITION 4,4:PRINT "Toon: ";:INPUT TOON
30 FOR VER=0 TO 14 STEP 2
40 POSITION 4,7:PRINT "Vervorming: ";VER
50 SOUND 0,TOON,VER,15
60 FOR W=1 TO 50:NEXT W
70 NEXT VER
80 SOUND 0,0,0,0
90 END
```

In regel 30 tot 70 worden alle vervormingsfactoren van de toon die na het RUN-commando wordt gevraagd, ten gehore gebracht. Zo kunnen we zelf nagaan hoe een toonhoogte bij verschillende vervormingsfactoren klinkt.
Probeer zelf eens een programma te maken dat om een vervormingsfactor vraagt en dat dan voor alle toonhoogten van 255 tot 0 laat horen wat het effect is.
Met behulp van deze programma's kan men in ieder geval een idee krijgen van de geluiden die de Atari met een geluidsgenerator kan produceren.

In het volgende programma gebruiken we de vervormingswaarde om het geluid van een overvliegend vliegtuig na te bootsen. Als het vliegtuig nadert neemt de geluidssterkte toe en de 'toonhoogte' van het lawaai neemt iets af. Als het vliegtuig recht boven de waarnemer is dan heeft de geluidssterkte een (oorverdovend) maximum aangenomen en neemt de toonhoogte snel af. In werkelijkheid wordt dit effect veroorzaakt door het zogenaamde Doppler-effect. Daarna vliegt het vliegtuig weer van ons weg en het geluid sterft uit.

```
10 REM OVERVLIEGEND VLIEGTUIG
20 FOR X=16 TO 30
30 SOUND 0,X,8,X-16
40 FOR W=1 TO 150:NEXT W
50 NEXT X
```

```
60 FOR X=30 TO 50
70 SOUND 0,X,8,15
80 FOR W=1 TO 70:NEXT W
90 NEXT X
100 FOR X=50 TO 64
110 SOUND 0,X,8,64-X
120 FOR W=1 TO 150:NEXT W
130 NEXT X
140 END
```

Wijsjes

Aan de hand van ons eerste programma hebben we op het Atari-orgel zelf een wijsje gespeeld. We kunnen echter ook de computer zelf laten spelen. We moeten de geluidsgeneratoren dan op vaste tijden steeds een andere toonhoogte laten aannemem.
Het volgende programma speelt een melodie. We maken in dit programma gebruik van de initialisatie van de tonen zoals die in het Atari-orgel-programma voorkwam.
We slaan de tonen op in strings. Daartoe reserveren we voor elke notenbalk een string. Een string voor de melodie en een string voor de begeleiding. Als er met akkoorden moet worden gespeeld kunnen we natuurlijk meer strings nemen en wel tot maximaal vier. In deze strings komen dan de te spelen tonen te staan (zie regel 300-320). In regel 410 wordt een toon van de eerste string (melodie) gespeeld en in regel 420 die van de begeleiding. We zien dat als we op de plaats van een noot een punt typen, de vorige noot wordt aangehouden. Op deze manier kunnen we dus tonen maken die langer duren. Als we tweemaal dezelfde toon willen aanslaan moeten we deze toon twee keer in de string zetten. In regel 430 is een wachtlus opgenomen om de tonen een tijdje te laten klinken. We kunnen door het getal in deze FOR...NEXT-instructie te veranderen het tempo van de muziek regelen. Als de muziek, die we de computer willen laten spelen erg lang wordt kunnen we de string erg lang maken door er steeds een stuk bij aan te plakken, of door een string met behulp van DATA- en READ-instructies steeds weer in te lezen.
N.B. In regel 320 ziet men weer een aantal CONTROL-karakters. Elke letter uit die regel waar een 'dakje' (^) voor staat moet als CONTROL-karakter worden ingevoerd. Dit houdt in dat we eerst de ESC-toets moeten indrukken en vervolgens de CONTROL-toets samen met het betreffende karakter.

```
10 REM SLAAP KINDJE SLAAP
20 REM INITIALISEER TONEN
30 DIM TOON$(256)
40 TOON$=CHR$(0)
50 TOON$(256)=TOON$
60 TOON$(2)=TOON$
70 FOR I=1 TO 21
80 READ J,X,Y
90 TOON$(J,J)=CHR$(X)
```

```
100 J=J+128:TOON$(J,J)=CHR$(Y)
110 NEXT I
120 DATA 3,243,230,4,217,204,5
130 DATA 193,0,6,182,173,7,162
140 DATA 153,1,144,136,2,128,0
150 DATA 67,121,114,68,108,102,69
160 DATA 96,0,70,91,85,71,81
170 DATA 76,65,72,68,66,64,0
180 DATA 99,60,57,100,53,50,101
190 DATA 47,0,102,45,42,103,40
200 DATA 37,97,35,33,98,31,29
299 REM DE MUZIEK IN STRINGS
300 DIM MUZ1$(57),MUZ2$(57)
310 MUZ1$="A..G.GF....Ac.cB.BA....Ac.ccBAB..B..AAAA
GFG..G..A..G.GF.."
320 MUZ2$="^F^AC^BC^B^AC^B^F.H^F^AC^G.C^F^AC^F.
H^A^G^F^A^G^F^G^A^BC..^F^A^BC^B^AC..C..^F^AC^BC
^B^A.."
399 REM SPEEL DE MUZIEK
400 FOR I=1 TO LEN(MUZ1$)
410 IF MUZ1$(I,I)<>"." THEN SOUND 0,0,0,0:SOUND
0,ASC(TOON$(ASC(MUZ2$(I,I)))),10,15
420 IF MUZ2$(I,I)<>"." THEN SOUND 1,0,0,0:SOUND
1,ASC(TOON$(ASC(MUZ2$(I,I)))),10,6
430 FOR W=1 TO 80:NEXT W
440 NEXT I
450 END
```

14. Over het scherm en over kleuren

Een van de zaken die een computer van een calculator onderscheidt is het feit dat we met een computer allerlei plaatjes op het beeldscherm kunnen weergeven. Deze mogelijkheid komt uiteraard bij zeer veel praktische toepassingen van pas: denk maar eens aan de grafische weergave van resultaten en afbeeldingen van figuren bij onderwijskundige toepassingen.
Ook zijn bij veel spelletjes plaatjes onontbeerlijk. Wie tegenwoordig langs de automatenhallen loopt, ziet ongetwijfeld de vele machines waar bijvoorbeeld raketten gillend over het beeldscherm worden afgeschoten. Allemaal zaken die zonder de computer van nu niet mogelijk zouden zijn.
De Atari biedt eveneens de mogelijkheid om plaatjes 'full color' weer te geven. In folders spreekt men vaak van 'graphics' als men het heeft over het afbeelden van figuren, en het feit dat ook bewegende patronen kunnen worden weergegeven wordt aangeduid met 'animated graphics'.

De modes

Het scherm is bij de Atari-computer in te stellen in verschillende modes. Mode 0, 1 en 2 zijn tekstmodes, dat wil zeggen dat we ze gebruiken om tekst weer te geven. Mode 0 is de normale mode, we zien deze na het inschakelen van de computer. Er passen dan 24 regels van 40 tekens op het scherm. Als we het proberen blijken we er slechts 24 regels van 38 tekens op te krijgen. Dit komt omdat bij het inschakelen automatisch de linkermarge op twee, en de rechtermarge op 38 wordt gezet. Later zullen we zien hoe we dit kunnen veranderen. We kunnen in deze mode de tekens van het alfabet, zowel de hoofdletters als de kleine letters, gebruiken, evenals de cijfers en diverse tekens als punt en komma.
Ook een hele set van grafische tekens kunnen we in mode 0 gebruiken. Een complete lijst van alle tekens is gegeven in appendix 1.
Mode 1 en 2 gebruiken we om grotere tekens te laten zien, eventueel in kleur. We kunnen dan tekens in totaal vier verschillende kleuren laten zien. Hier kunnen we echter slechts de helft van alle tekens tegelijk laten zien, en geen inverse video gebruiken.
De modes 3 tot en met 8 zijn grafische modes. Dat wil zeggen dat het scherm is opgedeeld in een matrix van puntjes, die we pixels noemen. Elk van deze pixels kunnen we een kleur en een lichtsterkte geven. Hier-

door kunnen we plaatjes maken. Het aantal pixels op het scherm bepaalt hoe fijn de tekening kan zijn die we kunnen maken, ofwel hoe groot de resolutie is.
Mode 3 geeft grote pixels: er zijn slechts 20 rijen van 40 pixels bruikbaar. Dit geeft een zeer lage resolutie. Mode 4 en 5 geven al een betere resolutie en in mode 6 en 7 kunnen we al een aardig plaatje maken. De modes 4 en 5, evenals 6 en 7, verschillen hierin dat we in de eerstgenoemden slechts twee kleuren kunnen gebruiken en in de modes 5 en 7 vier. In mode 8 kunnen we slechts één kleur in twee lichtsterkten gebruiken, maar deze levert dan wel een hoge resolutie. We kiezen een mode met het commando GRAPHICS, bijvoorbeeld met:

```
GRAPHICS 2
```

kiezen we mode 2. Op het gebruik van de verschillende modes komen we later terug, we gaan nu eerst eens kijken welke de kleurmogelijkheden van onze Atari-computer zijn.

De kleuren

Tot nu toe hebben we wellicht alleen een blauw scherm van de Atari gezien. De kleur van het scherm kunnen we echter veranderen. Typ maar eens in:

```
SETCOLOR 2,3,4
```

We zien nu dat het scherm rood wordt. Bij de opdracht SETCOLOR behoren drie getallen, die aangeven welk kleurregister we willen veranderen, welke kleur we willen gebruiken en ten slotte hoe de lichtsterkte van die kleur moet zijn.
De Atari-computer heeft vijf kleurregisters, genummerd van 0 tot en met 4, die elk kleurinformatie bevatten omtrent de te gebruiken kleur en lichtsterkte. Elk register heeft zijn eigen functie, register 2 bijvoorbeeld bepaalt in mode 0 (normale situatie) de kleur en lichtsterkte van de achtergrond. Dit register hebben we met bovenstaande SETCOLOR-opdracht op kleur 3 (rood-oranje) en lichtsterkte 4 gezet. Later zullen we zien voor welke functies de overige kleurregisters worden gebruikt. Voor de kleuren kunnen we kiezen uit het hieronder gegeven lijstje. Er zijn zestien kleuren, genummerd van 0 tot 15. Dit zijn de kleurcodes die we bij de SETCOLOR-opdracht moeten gebruiken. De lichtsterkte kunnen we instellen met een even getal vanaf 0, wat een zeer donker beeld produceert, tot 14, hetgeen een zeer licht beeld geeft.

Kleuren	Kleurcodes
grijs	0
licht oranje (goud)	1
oranje	2
rood-oranje	3
rose	4
paars	5

Kleuren	Kleurcodes
paars-blauw	6
blauw	7
blauw	8
lichtblauw	9
turquoise	10
groen-blauw	11
groen	12
geel-groen	13
oranje-groen	14
licht oranje	15

Met het volgende programmaatje kunnen we alle kleuren en kleurlichtsterkten laten zien:

```
10 FOR KLEUR=0 TO 15
20 FOR STERKTE=0 TO 14 STEP 2
30 SETCOLOR 2,KLEUR,STERKTE
40 FOR W=1 TO 50:NEXT W
50 NEXT STERKTE
60 NEXT KLEUR
70 SETCOLOR 2,9,4
80 END
```

De kleurregisters zijn zonder dat we daar iets voor hoeven te doen al op een kleur ingesteld. We weten dat dit voor register 2, die de kleur van de achtergrond in mode 0 bepaalt, blauw is. Hieronder staan de registers met hun standaard kleur- en lichtsterkte-instellingen.

Kleurregister	Kleur	Lichtsterkte
0	2 (oranje)	8
1	12 (groen)	10
2	9 (lichtblauw)	4
3	4 (rose)	6
4	0 (grijs)	0 (zwart)

De functies van de kleurregisters zijn voor de verschillende modes helaas anders. Zoals gezien diende register 2 voor de achtergrond in mode 0. Register 4 bepaalt de kleur en lichtsterkte van de rand van het scherm, om het eigenlijke rechthoekige scherm heen. Normaal is dit zwart, maar we kunnen dit met:

```
SETCOLOR 4,5,6
```

paars maken. In appendix 4 is weergegeven waarvoor de kleurregisters worden gebruikt in de verschillende modes. In sommige modes worden alle kleurregisters gebruikt, in andere modes zijn er slechts een aantal nodig. In die tabel is eveneens aangegeven hoe de diverse kleuren kunnen worden gebruikt.

15. Over PEEK, POKE en grote letters

POKE

In hoofdstuk 1 werd behandeld hoe het geheugen van de computer is ingedeeld in laden die ook wel eens worden aangeduid met de termen 'woord' of 'geheugenregister'. Iedere lade bevat acht vakjes waarin we de tekens 1 en 0 kunnen opslaan. Iedere geheugenplaats kan de getallen 0 tot en met 255 bevatten en wel in de zogenaamde binaire notatie. Het getal 0 komt overeen met een woord dat met allemaal nullen is gevuld en het getal 255 komt overeen met een woord waarvan elk vakje een 1 bevat (zie hoofdstuk 17).
Met de POKE-instructie kunnen we op een willekeurige plaats in het geheugen een getal opslaan. De vorm van de POKE-instructie is als volgt:

POKE geheugenadres,getal

We geven een voorbeeld:

```
10 PRINT "LINKERKANTLIJN (0-39) ";
20 INPUT LK
30 PRINT "RECHTERKANTLIJN (0-39) ";
40 INPUT RK
50 POKE 82,LK
60 POKE 83,RK
70 LIST 10,80
80 END
```

Het eerste dat opvalt bij een programma waarin POKE-instructies voorkomen, is dat de leesbaarheid veel slechter is. Wat er precies bij zo'n POKE-instructie gebeurt, blijkt immers niet uit de POKE-instructie zelf. Laten we het programma maar eens stap voor stap doornemen. De regels 10-40 leveren geen problemen. Regel 50 toont de eerste POKE-instructie. Als gevolg van deze POKE-instructie wordt het getal dat aan LK is toegekend in geheugenplaats 82 opgeslagen. Nu komt het belangrijkste: geheugenplaats 82 is steeds bepalend voor de plaats van de linkerkantlijn. Deze geheugenplaats is alleen voor dit doel gereserveerd! Evenzo is geheugenplaats 83 voor een vast doel gereserveerd, namelijk om de plaats van de rechterkantlijn aan te geven.
Probeer het programma maar eens voor verschillende waarden uit.

PEEK

De PEEK-instructie zou men kunnen opvatten als een omgekeerde POKE-instructie. Met de POKE-instructie kunnen we een getal op een bepaalde geheugenplaats wegzetten. Welnu, met de PEEK-instructie kunnen we een bepaalde geheugenplaats uitlezen. De uitgelezen waarde wordt aan een variabele toegekend. De algemene vorm is:

variabele = PEEK(geheugenplaats)

Uit deze vorm kunnen we tevens opmaken dat het bij PEEK in feite om een functie gaat. We geven een voorbeeld.
Geheugenplaats 53279 geeft steeds aan welke besturingstoetsen zijn ingedrukt. Dit zijn de toetsen OPTION, SELECT en START. Dit kunnen we aan de hand van het volgende programma demonstreren.

```
10 GRAPHICS 0
20 POSITION 5,3
30 POKE 752,1
40 PRINT "BESTURINGSTOETSEN INGEDRUKT:"
50 CODE=PEEK(53279)
60 POSITION 5,5
70 C=7-CODE
80 IF C=0 THEN PRINT "GEEN";
90 IF C>3 THEN C=C-4:PRINT "OPTION ";
100 IF C>1 THEN C=C-2:PRINT "SELECT ";
110 IF C=1 THEN PRINT "START";
120 PRINT "                 "
130 GOTO 50
```

Dit programma toont dan steeds de besturingstoets(en) die zijn ingedrukt. Geheugenadres 752 bevat een waarde die bepaalt of de cursor al dan niet zichtbaar is (zichtbaar=0, onzichtbaar=1). In regel 30 maken we met een POKE-instructie de cursor onzichtbaar. Dit is nodig omdat we anders een 'rondflitsende' cursor op het scherm zien. Probeer het programma maar eens uit met weglating van regel 30.
Regel 50 leest met de PEEK-instructie de code voor de besturingstoetsen, en wel uit geheugenplaats 53279. Deze code is een getal tussen 0 en 7, volgens onderstaande tabel als een zogenaamd 3-bitspatroon (drie enen en/of nullen) kunnen opvatten. De waarden van de bits 0, 1 en 2 worden respectievelijk bepaald door de toetsen START, SELECT en OPTION. Als de betreffende toets is ingedrukt, is het betreffende bit 0 en anders is het 1.

Code	**Bits**	**Besturingstoetsen**		
	2 0 1	**OPTION**	**SELECT**	**START**
0	0 0 0	aan	aan	aan
1	0 0 1	aan	aan	uit
2	0 1 0	aan	uit	aan
3	0 1 1	aan	uit	uit
4	1 0 0	uit	aan	aan
5	1 0 1	uit	aan	uit
6	1 1 0	uit	uit	aan
7	1 1 1	uit	uit	uit

In appendix 6 zijn een aantal nuttige geheugenadressen opgenomen met hun betekenis. In de volgende paragraaf zullen we daar gebruik van maken.

De tekstmodes

We hebben het al even over tekstmodes gehad, maar we hebben hier nog niets van gezien. Dat gaan we nu goedmaken, typ maar eens in:

```
GRAPHICS 2
```

Het scherm wordt nu in twee stukken verdeeld. In het onderste gedeelte staat nu 'READY'. Het is een 'stukje' scherm van vier regels hoog. We noemen dit gedeelte het tekstvenster. Het bovenste gedeelte is het grootst. Hier komen de grotere tekens te staan. We noemen dit het grafische venster.
We gaan nu grotere tekens op het scherm zichtbaar maken door:

```
PRINT #6;"MODE 2 IS TEKSTMODE"
```

We zien nu in grote oranje letters de tekst bovenaan in het grafische venster verschijnen. De toevoeging '#6' aan de PRINT-opdracht is nodig om de computer te vertellen dat de tekst in het grafische venster moet worden gezet en niet in het tekstvenster. We geven nu nogmaals dezelfde opdracht maar nu met de tekst in kleine letters, dus:

```
PRINT #6;"mode 2 is tekstmode"
```

We zien nu de tekst (overigens nog steeds in hoofdletters) in groen. Het cijfer 2 blijft oranje. Om de overige kleuren te gebruiken kunnen we de tekst in hoofdletters en inverse video typen of in kleine letters en inverse video. De kleuren worden dus bepaald door het feit of het al dan niet om kleine letters gaat en of de mogelijkheid van inverse video is gebruikt. In onderstaande tabel is weergegeven hoe de relatie tussen hoofdletters/kleine letters en inverse video met het gebruikte kleurregister is:

Gewenste kleurregister	**Tekstvorm**
0 (oranje)	hoofdletters
1 (groen)	kleine letters
2 (blauw)	hoofdletters in inverse video
3 (rose)	kleine letters in inverse video

We moeten nu nog het cijfer '2' in het groen op het scherm zien te krijgen. We zoeken daartoe de 'groene 2' op in de kolom 'standaard karakterset' van appendix 1. De bijbehorende ASCII-waarde blijkt 18 (decimaal) te zijn, dus het grafische teken CONTROL-R (we noteren ^R) zoals uit de kolom 'mode 0' blijkt. Als we nu de 2 vervangen door ^R dan krijgen we een groene 2. Probeer nu het volgende programma, waarin de tekens

die in inverse video moeten worden getypt tussen rechte haken zijn geplaatst.

```
10 GRAPHICS 2
20 PRINT #6;"MODE 2 MET 4 KLEUREN"
30 PRINT #6;"mode ^R met ^T kleuren"
40 PRINT #6;"[MODE 2 MET 4 KLEUREN]"
50 PRINT #6;[mode ^R met ^T kleuren]"
60 END
```

Voor het woord 'teken' gebruiken we ook wel de benaming 'karakter'. Een karakterset is dan een verzameling van karakters. De Atari-computer beschikt over drie karaktersets, afhankelijk van de mode. De karaktersets zijn weergegeven in appendix 1.
De grootste karakterset hebben we in mode 0 en deze bestaat uit grafische symbolen, hoofdletters, kleine letters, cijfers en leestekens. Bovendien kunnen al deze karakters ook nog invers worden weergegeven. De kleur wordt bepaald door kleurregister 2, de lichtsterkte door kleurregister 1. In de tekstmodes (mode 1 en 2) kunnen we kiezen uit een 'standaard karakterset' of een 'alternatieve karakterset'. De standaard karakterset bevat hoofdletters, cijfers en leestekens. Ieder karakter komt vier keer in de karakterset voor, steeds behorend bij een ander kleurregister. De kleur wordt bepaald door het kleurregister waartoe het karakter behoort. De alternatieve karakterset bevat de grafische symbolen en de kleine letters, eveneens verdeeld over vier kleurregisters.
De standaard karakterset en de alternatieve karakterset kunnen we niet tegelijk gebruiken. Normaal wordt de standaard karakterset gebruikt, maar met de opdracht:

```
POKE 756,226
```

kiezen we de alternatieve karakterset: geheugenlokatie 756 geeft namelijk aan waar de karakterset staat die we gebruiken. Normaal bevat geheugenplaats 756 de waarde 224. Als we er echter 226 in 'poken' dan wordt de alternatieve karakterset geselecteerd. Probeer dit maar eens, terwijl de uitvoer van het vorige programma nog op het scherm staat.
De hoofdletters zijn nu in kleine letters veranderd maar we zien nu hartjes op de plaatsen waar eerst niets stond. Dit wordt duidelijk als we in appendix 1 kijken. Het blijkt dan dat het hartje in de alternatieve karakterset overeenkomt met de spatie in de standaard karakterset. We zien nu dat er geen cijfers mogen worden gebruikt. De opmerkende lezer zal opvallen dat zowel in de oranje als in de groene regel de spaties als oranje hartjes worden weergegeven, terwijl ze in de blauwe en rose regel blauw zijn. Dit komt omdat in het programma spaties en inverse spaties zijn gebruikt. In de normale tekstmodes zie je dit verschil niet. De vraag is nu natuurlijk of we nog wel spaties kunnen maken. Dat kan, als we een trucje toepassen. We kunnen namelijk de blauwe kleur van kleurregister 2 veranderen in zwart door de lichtsterkte met de SETCOLOR-instructie 0 te maken. Alle blauwe karakters zijn nu niet meer zichtbaar en kunnen dus als spatie worden gebruikt. We kunnen weer teruggaan naar de gewone karakterset met de opdracht:

```
POKE 756,224
```

Tot nu toe moesten we bij het gebruiken van een andere kleur in de tekstmode in appendix 1 het juiste teken opzoeken. We kunnen de tekens in een andere kleur echter ook 'berekenen' uit de originele tekens. Dit gaat met behulp van de ASCII-waarde van deze tekens. Als we bijvoorbeeld 32 optellen bij de ASCII-waarde van de letter A, dan krijgen we een groene A:

```
PRINT #6;CHR$(65);CHR$(65+32)
```

Welk getal er bij moet worden opgeteld is afhankelijk van de ASCII-waarde van het karakter en het kleurregister waartoe we het karakter willen converteren. De volgende conversies gelden voor de verschillende kleur-registers:

Te gebruiken conversie

ASCII-waarde	karakterset	kleurregister 0	1	2	3
0-31	alternatieve	32	0	160	128
32-63	standaard	0	-32	128	96
64-95	alternatieve	0	32	128	160
96-127	standaard	-32	0	96	128

Op deze tabel zijn twee uitzonderingen te geven. De puntkomma (ASCII-waarde 59) kan niet met kleurregister 3 (normaal rose) worden gebruikt. Het teken wordt dan opgevat als een RETURN (nieuwe regel). Hetzelfde geldt voor het ESCAPE-teken (ASCII-waarde 27). De rechte haak sluiten ']' (ASCII-waarde 93) evenals de kromme pijl naar rechtsboven (ASCII-waarde 125) betekent bij gebruik met kleurregister 1 (normaal groen) dat het scherm zal worden schoongemaakt. Alle overige tekens zijn in vier kleuren te gebruiken.

Het volgende programma drukt een tekst af in tekstmode 2 en wel in de gewenste kleur, hierin vindt de conversie in een subroutine plaats:

```
10 DIM TEKST$(100)
20 GRAPHICS 2
30 PRINT "Type een tekst: ";:INPUT TEKST$
40 PRINT "Kies een kleur (0-3)":INPUT KLEUR
50 GOSUB 100
60 GOTO 30
70 END
100 REM ### printstring in kleur ###
110 TK=ASC(TEKST$(1,1))
120 AL=(TK<32) OR (TK>95)
130 POKE 756,224+2*AL
140 KE=INT(KLEUR/2)=KLEUR/2
150 FOR TEKEN=1 TO LEN(TEKST$)
160 TK=ASC(TEKST$(TEKEN,TEKEN))
170 GR=(TK>63)
180 CON=32*(4*(KLEUR>1)+(1-2*GR)*((KE*AL)-(NOT(KE+AL))))
190 PRINT #6;CHR$(TK+CON);
```

```
200 NEXT TEKEN
210 RETURN
```

In de regels 30 en 40 wordt gevraagd naar een te printen tekst en naar de gewenste kleur. Dan wordt een subroutine aangeroepen, die de tekst moet converteren zodat de juiste kleur ontstaat.
In regel 110-120 van de subroutine wordt bepaald of het eerste teken van de string tot de normale of tot de alternatieve karakterset behoort. Dit gebeurt door de ASCII-waarde van dat teken te vergelijken met die van de alternatieve karakterset. In de regel die hier op volgt wordt de juiste karakterset gekozen.
In regel 140 wordt gekeken of het gewenste getal een 'even kleur' is. Dit is handig bij het converteren. Vervolgens wordt in een lus, voor alle tekens van de tekststring, de conversie uitgevoerd en het geconverteerde karakter wordt geprint. De eigenlijke conversie vindt in regel 180 plaats. Deze regel is erg ingewikkeld. Dit komt hoofdzakelijk doordat voor verschillende karaktersets en verschillende kleuren de conversie ook verschillend is.

Functies voor tekstmodes

POSITION
De POSITION-instructie zet de cursor op een zelf te kiezen plaats. Als we nu in een tekstmode de POSITION-instructie gebruiken dan heeft deze op het grafische scherm betrekking en niet op het tekstvenster. Dit ondanks het feit dat er op het grafische scherm in een tekstmode geen cursor te zien is.

LOCATE
Met de LOCATE-instructie kunnen we opvragen wat er op het scherm staat. De LOCATE-instructie ziet er als volgt uit:

```
LOCATE X,Y,teken
```

Waarbij X,Y de coördinaten zijn van de plaats waar we willen kijken. De ASCII-waarde van het teken dat op deze plaats staat wordt aan de variabele TEKEN toegekend. In plaats van TEKEN mag natuurlijk ook een andere variabele worden gebruikt. De positie op het scherm waar vervolgens wordt geprint (dus eigenlijk de positie van de onzichtbare cursor) is na de LOCATE-instructie één plaats opgeschoven. In het volgende programma worden deze beide instructies gedemonstreerd. De tussen rechte haken geplaatste tekens moeten hierbij in inverse video worden getypt.

```
10 GRAPHICS 2
20 FOR I=1 TO 5*20
30 PRINT #6;CHR$(INT(RND(0)*26+65));
40 NEXT I
50 X=RND(0)*17
60 LOCATE X,2,TEKEN
70 PRINT #6;"[<<]"
```

```
80 POSITION 2,8
90 PRINT #6;"ASCII-GETAL: ";TEKEN
100 POSITION X,2
110 PRINT #6;CHR$(TEKEN+32)
120 POSITION 2,8
130 PRINT #6;"TEKEN IS : ";CHR$(TEKEN)
140 END
```

16. Over plaatjes

We gaan ons in dit hoofdstuk bezighouden met de grafische mogelijkheden van de Atari-computer. Met 'grafische mogelijkheden' doelen we dan op de mogelijkheid allerlei figuren op het scherm te produceren. We zullen zien dat de Atari hiertoe veel mogelijkheden biedt. We hebben al gezien dat de modes 3 tot en met 8 kunnen worden gebruikt om plaatjes te maken. Deze modes verschillen in resolutie, dat wil zeggen in fijnheid waarmee we figuren kunnen weergeven en in het aantal kleuren dat we mogen gebruiken. De onderstaande tabel geeft een overzicht: voor alle duidelijkheid, de term pixels heeft betrekking op de beeldpunten waarin het scherm is opgedeeld.

Mode	**Aantal pixels** hoog	breed	**Aantal kleuren**
3	20	40	4
4	40	80	2
5	40	80	4
6	80	160	2
7	80	160	4
8	160	320	1

De volgende instructies staan ons nu ter beschikking:

COLOR: Om aan te geven in welke kleur moet worden getekend.
PLOT: Voor het tekenen van allerlei figuren. In feite voor de weergave van een puntje op een aangegeven plaats. Door veel puntjes naast elkaar te tekenen kunnen we allerlei figuren samenstellen. Voor de lijn is een aparte instructie aanwezig.
DRAWTO: Voor het tekenen van een lijn.
XIO: Voor het inkleuren van een figuur.
LOCATE: Een functie om na te gaan welke kleur een bepaald pixel heeft.

COLOR

Deze instructie wordt gebruikt om aan te geven welke kleur de computer moet hanteren. Met de opdracht:

```
COLOR nr
```

selecteren we een kleur met nummer 'nr'. Het getal 'nr' mag bij de mode waarin vier kleuren zijn toegestaan 0, 1, 2 of 3 zijn. Bij de andere modes (1 of 2 kleuren toegestaan) mag 'nr' 0 of 1 zijn. Met COLOR 0 wordt altijd de achtergrondkleur bedoeld, dit is de kleur waarmee wordt begonnen, deze is zwart, behalve bij mode 8, dan is deze blauw.
Voordat we gaan tekenen moeten we dus een COLOR-opdracht geven, omdat we anders niets zien. Bij de kleurnummers 1-3, zoals we die bij de COLOR-opdracht gebruiken, horen de kleurregisters 0-2. Dus als we de kleur groen (kleurregister 1) willen gebruiken dan moeten we de opdracht:

```
COLOR 2
```

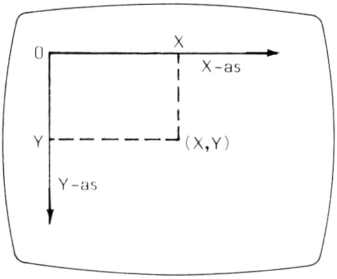
geven. Dit kan uiteraard alleen in de mode waarin vier kleuren zijn toegestaan. In appendix 4 is precies te vinden waarvoor de verschillende kleurregisters in de diverse modes worden gebruikt.

PLOT

De PLOT-instructie wordt gebruikt om allerlei plaatjes weer te geven. Met de PLOT-instructie kunnen we in feite alleen een punt op het scherm plaatsen en wel in een van tevoren met COLOR opgegeven kleur. Door nu veel punten naast elkaar te plaatsen kunnen we plaatjes construeren. De algemene vorm van de PLOT-instructie is:

```
PLOT x,y
```

Hiermee wordt dan een punt op de coördinaten x,y geplaatst. Het scherm is hierbij als volgt ingedeeld:

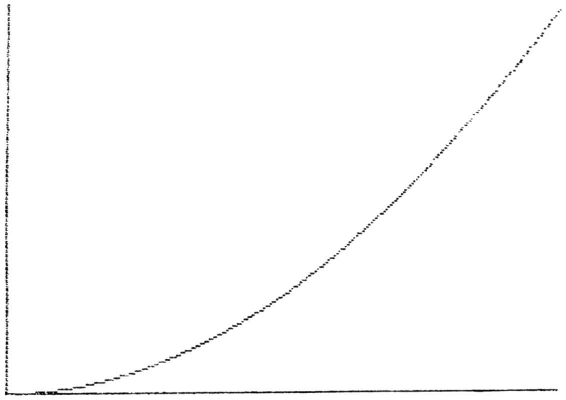


Het scherm is hier getekend voor mode 6 of 7. De coördinaten lopen van 0 tot 159 en van 0 tot 79, zodat we een totaal veld van 80 bij 160 pixels kunnen gebruiken. In het nu volgende programmaatje zien we hoe we met de PLOT-instructie een functie kunnen tekenen:

```
10 GRAPHICS 7
20 COLOR 1
30 PLOT 0,0
```

```
40 DRAWTO 0,79
50 DRAWTO 159,79
60 COLOR 2
70 FOR X=0 TO 159
80 Y=X^2/(159*159)*78
90 PLOT X,79-Y
100 NEXT X
110 END
```

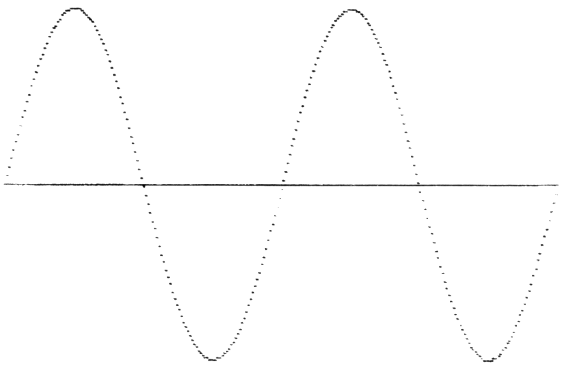
Resultaat:



In regel 20 wordt als kleur oranje gekozen. Met deze kleur tekenen we aan de hand van de regels 30, 40 en 50 een assenstelsel. De regels 70-100 vormen een FOR...NEXT-lus, waarin twee belangrijke instructies worden herhaald. Regel 80 berekent steeds een nieuwe functiewaarde, die dan in regel 90 op het scherm wordt gezet. Omdat we onze functie laten beginnen in de linker onderhoek, hebben we bovendien een correctie toe moeten passen. De computer rekent immers steeds vanuit de linker bovenhoek.
Het volgende voorbeeld toont een programma waarmee een sinusfunctie wordt afgebeeld. We moeten in dit programma weer dezelfde correctie gebruiken. Het programma luidt:

```
10 GRAPHICS 7
20 COLOR 1
30 PLOT 0,39
40 DRAWTO 159,39
50 COLOR 2
60 DEG
70 FOR X=0 TO 159
80 Y=39*SIN(X/159*2*360)
90 PLOT X,39-Y
100 NEXT X
110 END
```

Resultaat:



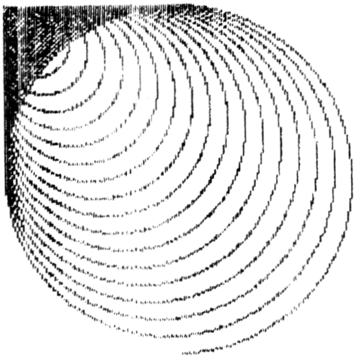
Het programma is 'rechttoe rechtaan' opgezet en zal dus geen problemen opleveren. In regel 10 wordt eerst weer het grafische scherm gedefinieerd. Daarna wordt in regel 20-40 een horizontale as getekend (midden over het scherm). In de FOR...NEXT-lus die in de regels 70-100 staat worden de juiste waarden berekend en getekend.
Uiteraard kunnen we met de PLOT-instructie ook cirkels construeren en wel aan de hand van de bekende formules:

x-coördinaat = r*cos(fi)
y-coördinaat = r*sin(fi)

Door bij x en y een constante waarde op te tellen, kunnen we het middelpunt van de cirkel verplaatsen. Het nu volgende programma gaat van deze instructies uit. De waarde K die in het programma staat heeft tot gevolg dat de straal en het middelpunt van de cirkel niet constant blijven. Typ het programma maar eens in en kijk wat er gebeurt. Het resultaat is stellig verbluffend...
Programma:

```
10 GRAPHICS 8
20 COLOR 1
30 DEG
40 FOR K=0 TO 20*360
50 R=79*(1-K/360/20)
60 X=R+R*COS(K)
70 Y=R+R*SIN(K)
80 PLOT X,Y
90 NEXT K
100 END
```

Resultaat:



Ten slotte merken we hierbij op dat we een punt kunnen wissen en wel door het met COLOR 0 nogmaals te plotten.

DRAWTO

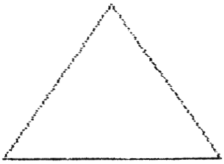
De Atari-computer biedt de mogelijkheid om direct rechte lijnen aan te geven. We doelen hierbij op de DRAWTO-instructie. Bij de constructie:

```
PLOT a, b
DRAWTO x,y
```

wordt een lijn van punt (a,b) naar punt (x,y getrokken. Na een DRAWTO-instructie kan weer een volgende DRAWTO-instructie worden geplaatst, enz. Het beginpunt van een nieuwe lijn is dan steeds het eindpunt van de vorige lijn. Dit wordt aan de hand van het volgende programma geïllustreerd. Programma:

```
10 GRAPHICS 7
20 COLOR 2
30 PLOT 0,70
40 DRAWTO 100,70
50 DRAWTO 50,10
60 DRAWTO 0,70
70 END
```

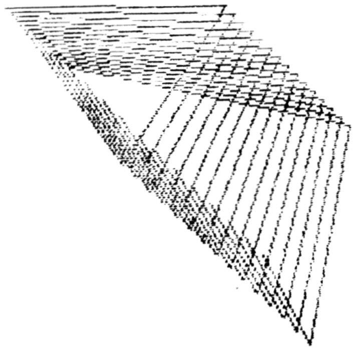
Resultaat:



Als gevolg van regel 30 en 40 wordt een horizontale lijn getrokken. Hierna volgt de lijn van het rechter hoekpunt tot de top van de driehoek (regel 50) en hierna de lijn van de top tot de linkerhoek (regel 60).
Het volgende programma toont een aardige variatie op het vorige programma:

```
10 GRAPHICS 8
20 COLOR 1
30 FOR K=1 TO 30 STEP 2
40 PLOT K,K
50 DRAWTO 2*K+100,2*K
60 DRAWTO 3*K+50,3*K+50*SQR(2)
70 DRAWTO 4*K,4*K
80 NEXT K
90 END
```

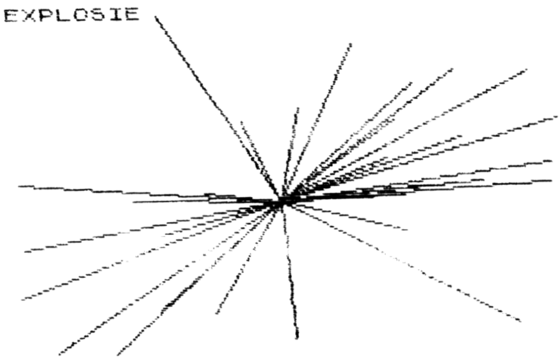
Resultaat:



Bij het nu volgende programma wordt iedere lijn vanuit het punt 79,39 getrokken (regel 50). De plaats waar de lijn naar toe wordt getrokken, wordt met de functie RND bepaald. We krijgen een verrassend leuk resultaat. Programma:

```
10 GRAPHICS 7
20 PRINT "EXPLOSIE"
30 FOR K=1 TO 30
40 COLOR INT(RND(0)*3+1)
50 PLOT 79,39
60 X=RND(0)*159
70 Y=RND(0)*79
80 DRAWTO X,Y
90 NEXT K
100 END
```

Resultaat:



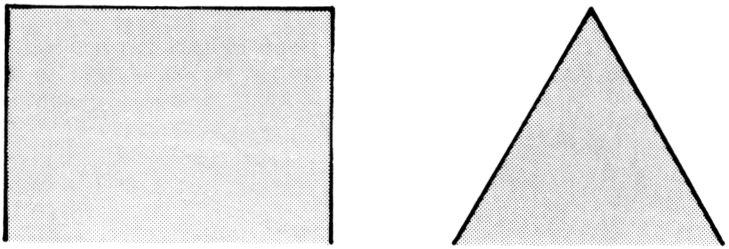
XIO

De Atari-computer kent een erg handige instructie om vlakken te kunnen inkleuren. Dit is de XIO-opdracht. Deze vult namelijk het vlak, waarvan we van tevoren de randen hebben opgegeven. We bekijken dit aan de hand van het volgende voorbeeld; een rechthoek en een driehoek worden ingekleurd. Programma:

```
10 GRAPHICS 7
20 PRINT "RECHTHOEK","DRIEHOEK"
30 COLOR 2
40 POKE 765,1
50 PLOT 60,50
60 DRAWTO 60,20
70 DRAWTO 20,20
80 POSITION 20,50
90 XIO 18,#6,0,0,"S:"
100 PLOT 140,50
110 DRAWTO 120,20
120 POSITION 100,50
130 XIO 18,#6,0,0,"S:"
140 END
```

Het resultaat van dit programma is op de volgende pagina afgebeeld. In regel 30 bepalen we de kleur van de figuren, dit betreft echter alleen de rand. Vervolgens 'poken' we de kleur waarmee moet worden opgevuld in adres 765. Dit gegeven gebruikt de XIO-instructie later. Nu moeten we de figuur tekenen. In regel 50-70 tekenen we eerst de rechterzijde en dan de bovenzijde van de vierhoek (deze volgorde is belangrijk). Vervolgens geven we met de POSITION-instructie aan waar de linker onderhoek zich bevindt. In regel 90 wordt dan uiteindelijk de rechthoek opgevuld met kleur 1. De XIO-opdracht die hiervoor zorgt ziet er als volgt uit:

```
XIO 18, 6,0,0,"S:"
```



RECHTHOEK DRIEHOEK

Resultaat van het inkleurprogramma van pagina 125.

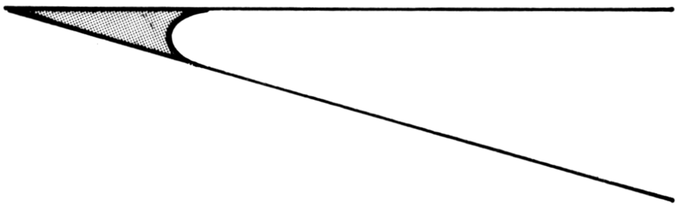
XIO 18 geeft aan dat het om de 'inkleuren-instructie' gaat. De waarden #6 en "S:" geven aan dat de bewerking op het beeldscherm plaats moet vinden en de twee nullen zijn zogenaamde dummy-waarden (dat wil zeggen: hebben geen betekenis). Bij de driehoek tekenen we eerst de rechterzijde (regel 100-110). Vervolgens geven we de linker onderhoek aan, waarna aan de hand van de XIO-instructie de driehoek wordt ingekleurd.

Hoe werkt XIO?
Om te begrijpen hoe de XIO-instructie nu wel precies werkt moeten we twee punten in ogenschouw nemen. Ten eerste moeten we denken aan het punt waar het laatst geplot werd, met andere woorden waar de laatste PLOT-instructie gegeven werd of waar de laatste DRAWTO-instructie eindigde (we noemen dit even het beginpunt).
Ten tweede moeten we het punt in ogenschouw nemen dat we met de POSITION-instructie hebben aangegeven (eindpunt). De XIO-instructie trekt nu een lijn van het beginpunt naar het eindpunt en wel in de kleur die met de laatste COLOR-instructie werd aangegeven. We kunnen dit zien aan de linkerzijden van rechthoek en driehoek: deze zijn groen. De figuur wordt opgevuld door vanaf het beginpunt tot het eindpunt steeds horizontale lijnen te trekken en wel in de met de POKE-instructie aangegeven kleur. Het trekken van zo'n lijn stopt bij het kruisen van een andere lijn of bij een punt. Bij het gebruik van de XIO-instructie ontstaat dus een gekleurd vlak met een horizontale boven- en onderkant en een rechte linkerzijkant. De rechterzijkant kan vele vormen aannemen zoals we in het volgende programma zullen zien.

```
10 GRAPHICS 7
20 SETCOLOR 0,3,4
30 COLOR 1
40 PLOT 159,79
50 FOR Y=78 TO 1 STEP -1
60 X=758/(Y*5)+Y*1.9
70 PLOT X,Y
80 NEXT Y
90 DRAWTO 159,0
100 DRAWTO 0,0
110 POSITION 158,79
120 POKE 756,1
```

```
130 XIO 18,#6,0,0,"S:"
140 END
```

Resultaat:



De rechter zijkant wordt door regel 60 bepaald. Het is een hyperboolachtige vorm.
Als bij het inkleuren geen kruisende lijn wordt gevonden, dan wordt verder gegaan op de volgende lijn. We kunnen, als we linker- en rechterzijkant verwisselen, ook alles buiten een figuur inkleuren! Dit zien we in het volgende voorbeeld gebeuren.

```
10 GRAPHICS 6
20 SETCOLOR 0,3,4
30 COLOR 1
40 PLOT 0,79
50 DRAWTO 120,10
60 DRAWTO 0,0
70 DRAWTO 159,0
80 POSITION 0,79
90 POKE 765,1
100 XIO 18,#6,0,0,"S:"
110 END
```

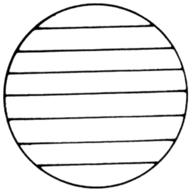
LOCATE

Met de LOCATE-instructie kunnen we, zoals we al gezien hebben, vanuit het programma 'kijken' naar wat er op het scherm staat. Resulteert deze functie bij tekstmodes in een ASCII-waarde, bij de grafische modes wordt een getal gevonden dat aangeeft welke kleur het aangewezen pixel heeft. Dit getal komt dan overeen met het getal van de COLOR-instructie waarmee dat punt werd geplot. Als ergens 'niets' staat levert de functie LOCATE een 0 op en anders een getal tussen 1 en 3.
In het nu volgende programma wordt daar handig gebruik van gemaakt. Eerst wordt een cirkel getekend (regel 50-100). Daarna worden steeds horizontale lijnen getrokken. Zodra een gekleurde lijn op het scherm met een andere lijn kruist zal de variabele KLEUR gelijk aan 1 worden (regel 160) en zal de lijn worden getekend (regel 170). Als weer een lijn wordt gekruisd, wordt KLEUR weer 0 en zal de rest van de horizontale

lijn niet meer worden getekend. Op deze wijze kan men bijvoorbeeld vlakken opvullen. We zullen dit uiteraard alleen doen als we een vlak met een niet-rechte linkerzijkant willen inkleuren, anders gebruiken we immers de XIO-instructie.
Programma:

```
10 GRAPHICS 7
20 PRINT "GLOBE"
30 COLOR 2
40 DEG
50 PLOT 79+39,39
60 FOR K=0 TO 360 STEP 5
70 X=79+39*COS(K)
80 Y=39+39*SIN(K)
90 DRAWTO X,Y
100 NEXT K
110 COLOR 1
120 FOR Y=25 TO 55 STEP 5
130 KLEUR=0
140 FOR X=0 TO 159
150 LOCATE X,Y,COL
160 IF COL<>0 THEN KLEUR=NOT KLEUR:X=X+1
170 IF KLEUR THEN PLOT X,Y
180 NEXT X
190 NEXT Y
200 END
```

Resultaat:



GLOBE

De LOCATE-instructie is ook een nuttige instructie als we bijvoorbeeld een botsing in een spelletje willen detecteren. Het programma kan daarop dan gepast reageren.

Nog iets over modes:

Modes 16 tot 24

In de modes 1 tot en met 8 is het scherm opgedeeld in een grafisch venster en een tekstvenster. Soms willen we het gehele scherm voor grafische toepassingen gebruiken. In dat geval verhogen we de mode

met 16. Hierbij moeten we er echter wel op letten, dat als het programma is afgelopen (of als er een fout optreedt) de tekst 'READY' verschijnt (resp. een foutmelding geprint wordt) en dat dan automatisch wordt overgeschakeld naar mode 0. We kunnen dan in ieder geval in plaats van:

```
90 END
```

bijvoorbeeld:

```
90 GOTO 90
```

aan het einde van het programma zetten. Maar bedenk wel dat we met de BREAK- of RESET-toets het programma onderbreken en dus in mode 0 terecht komen.

Geheugengebruik

Wat op het scherm wordt getoond komt in feite overeen met de inhoud van een gedeelte van het geheugen. Het zal dan ook duidelijk zijn dat voor de verschillende modes een verschillende hoeveelheid geheugen nodig is. Dit kan van belang zijn als we grotere programma's maken of grote arrays gebruiken. In appendix 5 is aangegeven hoeveel geheugen elke mode gebruikt. Bovendien staan in deze tabel de afmetingen van het scherm en de mogelijkheden ten aanzien van de kleuren.

17. Machine-instructies

Introductie

De opzet van dit boek is in eerste instantie om de lezer duidelijk te maken hoe men met behulp van de programmeertaal BASIC allerlei zaken op de computer kan uitvoeren.
Het programmeren met behulp van machine-instructies strookt nauwelijks met bovengenoemde doelstelling; niettemin zullen we de lezer een korte introductie in het programmeren in machine-instructies verschaffen, al was het alleen maar om te tonen wat 'programmeren in machine-instructies' nu wel inhoudt.
Machine-instructies, of zoals men ook wel zegt machine-codes, zijn de meest elementaire instructies die de computer kent; het zijn in feite de enige instructies die door de centrale verwerkingseenheid (zie hoofdstuk 1) worden uitgevoerd.
We zullen straks zien dat we machine-instructies met behulp van eenvoudige lettercodes aangeven. Daarna zullen we zien hoe we deze codes omzetten in hexadecimale getallen. In de meeste boekjes over het programmeren worden de machine-instructies door hexadecimale getallen weergegeven. We zullen tevens zien hoe we deze hexadecimale getallen in de ons vertrouwde decimale getallen kunnen omzetten, die we dan met behulp van POKE-instructies in het geheugen kunnen plaatsen.
Ten slotte moeten we bedenken dat in de afzonderlijke geheugenadressen alleen maar rijtjes enen en nullen kunnen worden geplaatst. Dit houdt in dat de decimale getallen die we door middel van POKE-instructies in de afzonderlijke geheugenlokaties plaatsen uiteindelijk in de vorm van rijtjes enen en nullen worden opgeslagen: we spreken dan van zogenaamde binaire getallen. Hier zullen we tevens dieper op ingaan.
Kort samengevat:

> Een programma in machine-instructies bevat instructies die met behulp van eenvoudige lettercodes worden omschreven;
>
> ↓
>
> met behulp van getallen worden deze lettercodes in zogenaamde hexadecimale getallen omgezet;
>
> ↓
>
> de hexadecimale tabellen worden in decimale getallen omgezet die dan door middel van POKE-instructies in het geheugen worden geplaatst. Uiteindelijk bevat het geheugen alleen maar series enen en nullen (binaire getallen) die met deze decimale getallen overeenkomen.

Hexadecimale getallen

Alvorens we ons betoog voortzetten geven we eerst een korte uiteenzetting rond hexadecimale getallen.
Hexadecimale getallen worden genoteerd met behulp van de volgende cijfers:

0 1 2 3 4 5 6 7 8 9

en de volgende letters:

A B C D E F

Deze letters stellen ook getallen voor. De onderstaande tabel toont voor iedere letter het getal dat er mee wordt aangegeven.

letter	waarde
A	10
B	11
C	12
D	13
E	14
F	15

Om onderscheid te maken tussen onze vertrouwde (decimale) getallen en de hexadecimale getallen plaatsen we voor de laatste categorie steeds het dollarteken ($).
Op grond van het bovenstaande volgt eenvoudig dat bijvoorbeeld 8=$8; of in woorden, ons decimale getal 8 wordt bij de hexadecimale getallen met $8 aangeduid.
Evenzo geldt:

10=$A
11=$B
12=$C
13=$D
14=$E
15=$F

Nog een voorbeeld:

2 + 10 = 12 *(decimaal)*

komt overeen met:

$2 + $A= $C *(hexadecimaal)*

Het is even wennen... maar moeilijk is het niet! Tot nu toe bespraken we hexadecimale getallen die uit één symbool (=cijfer) bestaan. In het hexadecimale stelsel kennen we echter ook getallen die uit meer cijfers bestaan, zoals:

$A23
en $21F

Hoe moeten we deze getallen nu 'vertalen' in onze bekende decimale getallen? Welnu, we gaan hierbij uit van de machten van 16.

macht	**notatie**	**waarde als decimaal getal**
16 tot de macht 0	16^0	1
16 tot de macht 1	16^1	16
16 tot de macht 2	16^2	16 x 16 = 256
16 tot de macht 3	16^3	16 x 16 x 16 = 4096
16 tot de macht 4	16^4	16 x 16 x 16 x 16 = 65536

Machten van 16.

Een getal als $A23 dient men als volgt om te zetten:

```
$ A 2 3
  | | |
  | | └── 3 x 16^0 =  3 x   1 =    3
  | |
  | └──── 2 x 16^1 =  2 x  16 =   32
  |
  └──── 10 x 16^2 = 10 x 256 = 2560
                              ----
            totaal            2595
```

We zien hoe de positie van ieder cijfer in het hexadecimale getal bepalend is voor de macht van 16 waarmee we dit getal vermenigvuldigen. Het meest rechtse cijfer correspondeert met 16^0, het getal hiernaast correspondeert met 16^1 enz.

Moeilijk... geen nood, BASIC komt ons te hulp. Het nu volgende programma kunnen we gebruiken om een hexadecimaal getal in een decimaal getal om te zetten.

```
10 DIM A$(5)
20 G=0
30 PRINT "HEXADECIMAAL GETAL= "
40 INPUT A$
50 L=LEN(A$)
60 FOR K=1 TO L
70 M=16^(L-K)
80 V=ASC(A$(K))
90 IF V>=65 AND V<=70 THEN W=V-55
100 IF V<=57 THEN W=V-48
110 G=G+W*M
120 NEXT K
130 PRINT INT(G+0.5)
140 END
```

Bij dit programma voeren we de hexadecimale getallen (zonder het dollarteken) in.

Voorbeeld

Na het RUN-commando verschijnt:

```
HEXADECIMAAL GETAL= ?
```

Nu typen we in:

A23 *(gevolgd door RETURN)*

en onmiddellijk verschijnt:

2595

Ook de omzetting van een decimaal naar een hexadecimaal getal kunnen we aan de hand van een programma uitvoeren.
Het volgende programma kan men gebruiken:

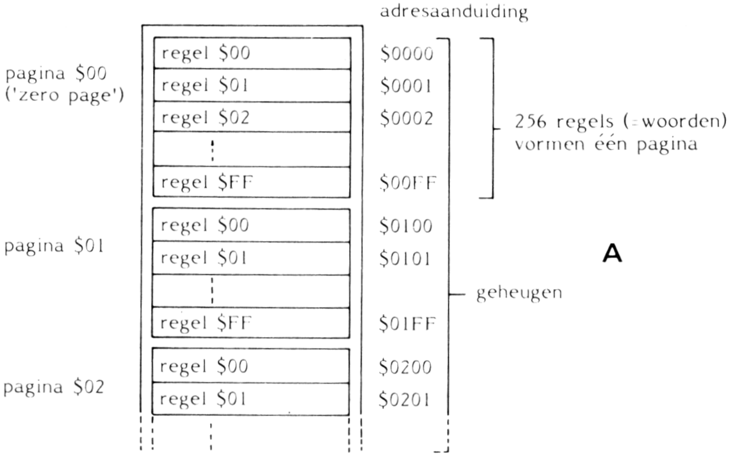
```
10 PRINT "GETAL= ";
20 INPUT A
30 B1=INT(A/(16*16*16))
40 R1=B1*16*16*16
50 A=A-R1
60 B2=INT(A/(16*16))
70 R2=B2*16*16
80 A=A-R2
90 B3=INT(A/16):R3=B3*16
100 B4=A-R3
110 PRINT B1,B2,B3,B4
120 END
```

Het resultaat verschijnt steeds in de vorm van vier cijfers 0 t/m 15. Met behulp van de tabel op pagina 131 dient u dan nog de cijfers 10 t/m 15 in de letters A t/m F om te zetten, of nog beter: u breidt het programma dienovereenkomstig uit!
Hiermee is onze bespreking van hexadecimale getallen afgerond.

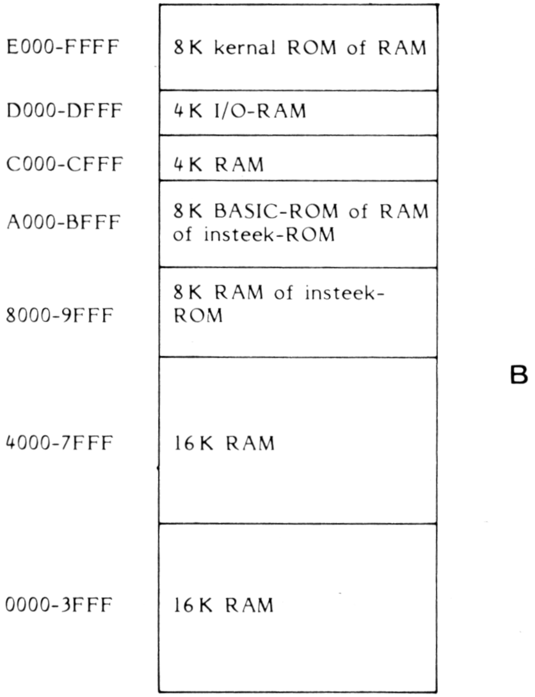
Geheugenadressen

Bij machine-instructies wordt veelvuldig naar bepaalde adressen in het geheugen verwezen. Er worden bepaalde getallen in bepaalde geheugenadressen geplaatst en andersom worden er getallen vanuit bepaalde geheugenadressen 'opgehaald'. We zullen daar straks meer in detail op ingaan. De geheugenadressen worden in een programma aangegeven met behulp van hexadecimale getallen bestaande uit vier cijfers. Volgens afspraak wordt het geheugen ingedeeld in porties van 256 woorden. Ter herinnering, ieder woord is een 'lade' van het geheugen waarin we 8 bits kunnen opslaan (zie hoofdstuk 1). Deze porties noemen we voortaan pagina's en de afzonderlijke woorden duiden we aan met regels (zie afb. A op blz. 134).
We zien in afb. A hoe ieder woord van het geheugen wordt aangegeven met behulp van een hexadecimaal getal, bestaande uit vier cijfers, bijvoorbeeld:

$01FF



Schematische voorstelling van het geheugen.



Samenstelling van het geheugen van de Atari.

Sommige adressen worden gebruikt voor speciale doeleinden. In appendix 6 vinden we een overzicht van een aantal van deze adressen. Schema B geeft een overzicht van de geheugenindeling van de Atari-computer.

Meer informatie over de 6502-processor

De 6502-processor (de centrale verwerkingseenheid) vormt in feite 'de hersenen' van onze computer. Deze processor bevat zelf ook een aantal interne geheugens, die we voortaan met de term *register* zullen aanduiden. De belangrijke registers zijn in de volgende tabel aangegeven.

aanduiding register	**omschrijving**
ACCU	register waar de feitelijke bewerkingen plaatsvinden zoals: - laad inhoud van bepaald geheugenadres in ACCU; - plaats inhoud van ACCU in bepaald geheugenadres; - schuif het enen- en nullenpatroon in ACCU één plaats naar links of naar rechts op; - plaats in alle vakjes van het register een nul;
PC	register waarin een geheugenadres is aangegeven en wel het adres waar de machine-instructie staat die zal worden uitgevoerd. De letters PC slaan op '**P**rogram **C**ounter' (vertaald: 'programmateller').
P	het zogenaamde processor-statusregister. Dit register geeft bepaalde toestanden van de 6510 aan, bijvoorbeeld of bij een bewerking het resultaat 0 is opgetreden.
S	de zogenaamde 'stack pointer'. Dit register onthoudt het geheugenadres waar het programma na afhandeling van een subroutine moet worden voortgezet.
X en Y	registers om een bepaalde manier van adresseren mogelijk te maken; het zogenaamde 'geïndexeerd adresseren'.

Machine-instructies

Alvorens specifieke machine-instructies te bespreken gaan we eerst in op de achtergrond bij het programmeren in machine-instructies, of zoals het ook wel eens genoemd wordt: 'assembler-programming'. We moeten ons voorstellen dat een deel van het geheugen wordt gevuld met getallen. Deze getallen worden in het geheugen gerepresenteerd door middel van series enen en nullen: binaire getallen.
We moeten ons verder indenken dat deze getallen bepaalde machine-instructies voorstellen. In een aantal gevallen stellen deze getallen echter geen machine-instructies voor, maar gewoon getallen. Uit het programma zelf volgt of we met een machine-instructie dan wel met een getal te maken hebben. We kunnen ons best voorstellen dat u op dit moment denkt

'hoe is dat nu mogelijk?'. Aan de hand van een concreet voorbeeld zullen we straks zien hoe dit alles in zijn werk gaat. We zullen dan zien dat er allereerst een bepaald *startadres* wordt aangewezen. De inhoud van dit geheugenadres wordt door de computer altijd opgevat als een instructie. Gewoonlijk worden de instructies vanaf het startadres één voor één in de opeenvolgende geheugenadressen geplaatst. In deze volgorde zullen ze dan ook automatisch worden afgewerkt. We zullen echter ook zien dat er instructies zijn waarmee we zogenaamde sprongen kunnen aangeven.
In het nu volgende zullen we enkele instructies bespreken. We zullen zien dat er instructies zijn om de inhoud van een bepaald geheugenadres in de ACCU of het X- of Y-register te plaatsen. Andersom zijn er ook instructies waarmee we gegevens van ACCU, X- of Y-register in bepaalde geheugenadressen kunnen plaatsen.
Daarnaast zijn er instructies waarmee we op de inhoud van ACCU, X- of Y-register bewerkingen kunnen uitvoeren. Deze bewerkingen zijn in feite opvallend eenvoudig, zoals 'tel bij de inhoud 1 op' of 'schuif het enen- en nullenpatroon één plaats naar rechts op'. We zullen ook zien dat er instructies zijn, waarmee wordt aangegeven op welk geheugenadres het programma dient te worden voortgezet. Een overzicht van de verschillende instructies is te vinden in appendix 8.

Adresseermethoden

De meeste instructies hebben betrekking op gegevens die óf in het geheugen geplaatst moeten worden, óf vanuit het geheugen moeten worden opgehaald. Er bestaan veel methoden om bepaalde geheugenplaatsen aan te wijzen. We spreken dan van adresseermethoden. Vaak zijn die methoden voor een beginneling lastig te doorzien. Het is in ieder geval voortdurend opletten geblazen.
In het onderstaande bespreken we in het kort de verschillende adresseermethoden. De verschillende methoden worden gewoonlijk met Engelse benamingen aangegeven.

Immediate (onmiddellijk)

We illustreren deze adresseerwijze aan de hand van de LDA-instructie. De LDA-instructie geeft aan dat we een gegeven (getal) in de ACCU moeten plaatsen. De instructie zelf noemen we voortaan 'de operator' en het gegeven 'de operand'.
Welnu, de afspraak is dat bij een LDA-instructie, waarbij de immediate-adresseerwijze wordt gehanteerd, de operand staat in het geheugenadres dat volgt op het geheugenadres waarin de LDA-instructie staat. Deze wijze van adresseren wordt met behulp van het teken # aangegeven. Voorbeeld:

HEXADECIMAAL			MNEMONISCH		
pagina	regel	instructie	operator	operand	commentaar
00	00	A9	LDA	#	laad accu (immediate)
00	01	0E		$0E	met getal $0E
00	02	volgende instructie			

In dit schema komen wellicht een groot aantal vreemde zaken voor. We beginnen met de drie kolommen links. De eerste twee kolommen geven de geheugenadressen door middel van een pagina- en een regelnummer aan. De derde kolom toont hoe deze geheugenadressen gevuld zijn en wel in de vorm van hexadecimale getallen. Op de eerste geheugenplaats zien we het getal $A9 en dit getal is de hexadecimale code voor de instructie:

LDA met immediate adresseerwijze

De hexadecimale codes voor alle instructies vindt men in appendix 8. Volgens afspraak staat het getal dat in de ACCU geladen moet worden in de daaropvolgende geheugenplaats. In ons voorbeeld gaat het om het getal $0E.
In de drie kolommen rechts ziet men een zogenaamde mnemonische beschrijving van het programma. Het woord mnemonisch geeft aan: 'beschrijvingsmethode die we kunnen onthouden'.
De eerste kolom geeft de operator, met andere woorden de instructie, aan. De volgende kolom geeft de operand aan. Het teken # wil zeggen 'de operand is via de immediate adresseerwijze te vinden'. De kolom commentaar behoeft geen verdere toelichting.
Uit dit voorbeeld kan men twee belangrijke wetenswaardigheden putten:

a. de operand wordt door een hexadecimale code aangegeven. Uiteindelijk wordt deze code in het geheugen zelf als een binair getal, dat wil zeggen een getal met enen en nullen, geschreven.
In dit geval geeft de code A9 aan: LDA-instructie met immediate adresseerwijze.
Voor iedere instructie in combinatie met de gekozen adresseerwijze bestaat een unieke hexadecimale code, die in appendix 8 te vinden is.

b. de totale instructie inclusief het aangegeven getal nemen bij dit voorbeeld twee geheugenplaatsen in beslag. We zeggen 'de instructie expandeert over twee plaatsen'. In appendix 8 zien we tevens over hoeveel geheugenplaatsen iedere instructie expandeert. Na het uitvoeren van deze instructie weet de computer dat de volgende instructie op regel 02 te vinden is.

Absolute (absoluut)

De absolute adresseerwijze bespreken we aan de hand van de STA-instructie. De STA-instructie geeft aan dat de inhoud van de ACCU op een bepaalde plaats in het geheugen moet worden geplaatst. Bij de absolute adresseerwijze geven we direct (dat wil zeggen in de twee opvolgende geheugenplaatsen in het programma) het regelnummer en het paginanummer van de beoogde plaats aan. Voorbeeld:

HEXADECIMAAL			MNEMONISCH		
pagina	regel	instructie	operator	operand	commentaar
03	00	8D	STA	(abs)	laad inhoud ACCU op $034E
03	01	4E		regel	4E=regeladres
03	02	03		pagina	03=pagina-adres
03	03	volgende instructie			

Kennelijk geeft de hexadecimale code 8D de instructie 'STA met absolute adresseerwijze' aan.

Zero page (pagina 0)

Deze methode lijkt als twee druppels water op de vorige. We hoeven hier het pagina-adres niet op te geven: volgens afspraak gaat het altijd om de allereerste pagina, dat wil zeggen pagina 0 (zie ook afb. A op pagina 134).

HEXADECIMAAL			MNEMONISCH		
pagina	regel	instructie	operator	operand	commentaar
02	04	A5	STA	(Z-page)	laad ACCU in $0004
02	05	04		04	regel op pagina 0

Accumulator (ACCUM.)

Deze adresseerwijze heeft betrekking op de ACCU zelf. We geven als voorbeeld de ASL-instructie. Deze instructie geeft aan dat alle enen en nullen één plaats naar links moeten worden opgeschoven. In het meest rechtse hokje (bit) komt nu een nul te staan. De één of nul die in het meest linkse hokje stond wordt in een afzonderlijk register, dat maar uit één vakje (1 bit) bestaat, opgeslagen. Dit afzonderlijke register draagt de naam C en deze naam is afkomstig van 'Carry' (overdracht).

HEXADECIMAAL			MNEMONISCH		
pagina	regel	instructie	operator	operand	commentaar
04	10	0A	ASL	(ACCU)	schuif bits 1 plaats naar links

Implied (impliciet)

Hierbij kunnen we nauwelijks van een adresseerwijze spreken. Zo is er bijvoorbeeld de CLC-instructie. Deze geeft aan het register C (zie het voorafgaande) te wissen ('clear carry'). Hierbij heeft de instructie dus impliciet ('vanzelf') betrekking op het C-register.

Relative (relatief)

Deze adresseermethode komt neer op 'ga N geheugenplaatsen voor- of achteruit'. We behandelen dit bij de zogenaamde spronginstructies.

Indirect (indirect)

Een belangrijke adresseerwijze. De instructie strekt zich uit over drie geheugenplaatsen. De tweede en derde plaats geven een regelnummer en

een paginanummer van een geheugenadres aan, bijvoorbeeld 04 (regelnummer) en 12 (paginanummer). De afspraak is nu dat op de aangegeven plaats, dat wil zeggen voor het voorbeeld op plaats $1204 en op de daaropvolgende plaats (dat wil zeggen $1205) het regel- en paginanummer van de operand staat. We bereiken de operand dus via een tussenweg, vandaar de naam 'indirect'.

Absolute indexed; ABS,X en ABS,Y

Hierbij wordt gebruik gemaakt van de X- en Y-registers. Bij deze adresseerwijze strekt de instructie zich opnieuw over drie geheugenplaatsen uit. De tweede en derde plaats geven op de bekende manier een volledig adres aan. De afspraak is dan dat de inhoud van het X- of Y-register bij dit adres wordt opgeteld en zo wordt ten slotte het adres van de operand gevonden. Deze adresseerwijze is van belang als we een rij opeenvolgende adressen willen bereiken. Met de zogenaamde increment-instructie (INX en INY) verhogen we dan X of Y steeds met 1.

Zero page indexed; Z.PAGE.X en Z.PAGE.Y

Als de vorige adresseerwijze, maar nu wordt het paginanummer weggelaten. We gaan er dan van uit dat het paginanummer steeds 0 is.

Indexed indirect (IND,X) en indexed indirect (IND,Y)

Deze adresseerwijzen vallen buiten het kader van deze eerste introductie.

Sprongen

We kennen twee soorten sprongen, namelijk:
1. De onvoorwaardelijke sprong (JMP-instructie). Deze komt overeen met de GOTO-instructie in BASIC.
2. De voorwaardelijke of conditionele sprong (branch-instructie). Deze komt overeen met de IF... THEN regelnummer-instructie in BASIC.

JMP-instructie

Geeft een onvoorwaardelijke sprong aan. De sprong kunnen we zowel absoluut als indirect aangeven.

De branch-instructies

Alvorens we de constructie bespreken merken we op dat in het P-register (zie blz. 135) verschillende bits 1 of 0 kunnen worden, al naar gelang er bij een bewerking een bepaald resultaat optreedt. Dit P-register wordt daartoe opgedeeld in afzonderlijke vakjes met de namen:

naam	betekenis
N	negatief
V	overflow
C	carry
Z	zero
I	interrupt disable (buiten werking)
D	decimal mode
B	break command

Het voert ons veel te ver om op al deze begrippen afzonderlijk in te gaan. We bespreken het vakje met de naam Z.
De regel is, dat als het resultaat van een vorige bewerking de waarde 0 (zero) oplevert, in Z een 1 komt te staan. We zeggen wel eens 'de Z-vlag wordt geset'. Hierop kunnen we dan met een branch-instructie een test uitvoeren. Als het om het testen van de Z-vlag gaat gebruiken we de branch-instructie BEQ (zie appendix 8).
Iedere branch-instructie wordt gevolgd door een getal dat aangeeft hoeveel stappen moeten worden overgeslagen om de volgende instructie te vinden, tenminste als aan de voorwaarde is voldaan.

Voorbeeld

HEXADECIMAAL			MNEMONISCH			
adres	instr.		label	operator	operand	commentaar
0200	F0			BEQ		als Z=1 ga naar
0201	02				VERDER	VERDER
0202	. .	2 plaatsen				
0203	. .					
0204	. .		VERDER:	nieuwe instructie		
0205	. .					

De instructie BEQ wordt door F0 aangegeven. Hierna volgt 02, kortom als Z=1 worden twee plaatsen overgeslagen. We zien hoe we in het mnemonische programma labels hebben weergegeven om een en ander overzichtelijk weer te geven.
De andere belangrijke branch-instructies zijn:

branch-instructie	voorwaarde
BCC	C=0
BCS	C=1
BEQ	Z=1
BMI	N=1
BNE	Z=0
BPL	N=0
BVC	V=0
BVS	V=1

We zien in appendix 8 dat er ook speciale instructies zijn waarmee we

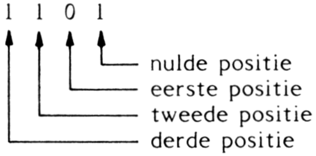
bewerkingen op C, Z, N en V kunnen uitvoeren (inhoud 1 of 0 maken). Als we een sprong terug willen aangeven moeten we in feite een negatief getal aangeven. We zullen aan het slot van dit hoofdstuk in het kort vermelden hoe dit in zijn werk gaat.

Subroutines

De JSR-instructie is de zogenaamde 'jump to subroutine-instructie'. De instructie lijkt als twee druppels water op de JMP-instructie, met dit verschil dat het terugkeeradres in de zogenaamde stack (stapelgeheugen) wordt bewaard. Terugkeer naar dit adres volgt als resultaat van de RTS-instructie. Dit alles verloopt automatisch, maar de ware programmeur gebruikt natuurlijk weer vele trucs. Een van die trucs is om gegevens in het terugkeeradres op te slaan die vanuit de subroutine worden opgevraagd (zogenaamde parameter-overdracht). We moeten er dan wel voor zorgen dat de stack een nieuw terugkeeradres bevat. We kunnen daartoe allerlei manipulaties met de stack uitvoeren. Zie in appendix 8 de instructies PHA, PHP, PLA en PLP.
Voorlopig is dit alles voor de beginner stellig iets te lastig. Het enige dat we goed moeten onthouden is het feit dat ieder machine-instructie-programma, dat vanuit een BASIC-programma wordt opgeroepen, eindigt met de instructie RTS!
Op deze wijze volgt dan na het afwerken van het machine-instructie-programma automatisch de terugkeer naar het BASIC-programma.

Binaire getallen

Ten slotte geven we nog enige informatie over binaire getallen. Dit is belangrijk met betrekking tot branch-instructies en rekenkundige bewerkingen.
Bij binaire getallen worden alleen de tekens 1 en 0 gebruikt. Hierbij geldt de regel dat de positie van een 1 met een macht van twee overeenstemt. De posities worden, evenals bij de hexadecimale (en decimale) getallen, weer vanaf de rechterkant geteld. We beginnen steeds met 0 te tellen. Bijvoorbeeld:



Dit getal stemt dan overeen met:

1 op nulde positie:	$1 \times 2^0 =$	1
1 op tweede positie:	$1 \times 2^2 =$	4
1 op derde positie:	$1 \times 2^3 =$	8
		13

De omzetting van een decimaal getal naar een binair getal geschiedt door steeds machten van 2 van het getal af te trekken, en wel steeds de grootst mogelijke macht.
Voorbeeld: omzetting van 13 in het bijbehorende binaire getal.

$$\begin{array}{lrl} & 13 & \\ 2^3 = & 8\ - & \text{(is de grootst mogelijke macht van 2)} \\ \hline & 5 & \\ 2^2 = & 4\ - & \text{(is de grootst mogelijke macht van 2)} \\ \hline & 1 & \\ 2^0 = & 1\ - & \text{(is de grootst mogelijke macht van 2)} \\ \hline & 0 & \end{array}$$

De getallen 2^3, 2^2 en 2^0 vormen weer het binaire getal 1101 (zie ook het voorafgaande).
We kunnen binaire getallen uiteraard ook optellen.

Voorbeeld 1

$$\begin{array}{rl} 1 & (=1 \text{ decimaal}) \\ 1\ + & (=1 \text{ decimaal}) \\ \hline 10 & (=2 \text{ decimaal}) \end{array}$$

Voorbeeld 2

$$\begin{array}{rl} 101 & (=5 \text{ decimaal}) \\ 111\ + & (=7 \text{ decimaal}) \\ \hline 1100 & (=12 \text{ decimaal}) \end{array}$$

Negatieve getallen

Ieder woord van het geheugen bevat slechts acht vakjes en dat levert natuurlijk beperkingen op. Allereerst spreken we af dat het meest linkse vakje ('most significant bit') het *teken* van een getal aangeeft. Als er een 0 in staat is het getal positief en als er een 1 in staat wordt het getal als een negatief getal opgevat.
De notatie van een negatief getal als een rijtje enen en nullen verkrijgt men eenvoudig door de volgende procedure toe te passen (zgn. 'two's complement code'):

a. ga uit van het positieve getal;
b. maak van iedere 0 een 1 en andersom van iedere 1 een 0;
c. tel er 1 bij op.

Voorbeeld: -18

Ad a.	ga uit van 18	00010010
Ad b.	keer om	11101101
Ad c.	+ 1	11101101
		00000001 +
		11101110

Zo zien we dat -18 als 11101110 wordt geschreven.

De afspraken zijn van groot belang als we met branch-instructies en/of met rekenkundige instructies werken, zoals ADC en SBC.

Voorbeeld met een branch-instructie

We geven nog een voorbeeld van een branch-instructie. Hierbij testen we of Z=1 en als dit zo is willen we vijf stappen *terugspringen* (-5).
We gaan uit van een reeks instructies die vanaf adres 02A1 in het geheugen is geplaatst.

HEXADECIMAAL			MNEMONISCH			
adres	inhoud		label	operator	operand	commentaar
02A1	. . .		START			. . .
02A2	. . .					. . .
02A3	. . .	Z=1				. . .
02A4	F0			BEQ		als Z=1 spring
02A5	FB				START	naar START
02A6	. . .					. . .

De instructie BEQ wordt door F0 aangegeven. De hexadecimale aanduiding FB geeft '5 stappen terug' aan, kortom het getal -5.
Hoe komen we nu aan de aanduiding FB?
Welnu, de waarde -5 vinden we volgens het zojuist besproken recept:

```
+5              00000101

omdraaien       11111010
+1              00000001  +
                --------
                11111011     (=-5)
```

De zo gevonden aanduiding 11111011 geeft dus -5 aan. Pas nu op: voor de hexadecimale notatie splitsen we dit getal altijd in twee gelijke stukken:

1111 ¦ 1011

1111
=15 (decimaal)
=F (hexadecimaal)

1011
=11 (decimaal)
=B (hexadecimaal)

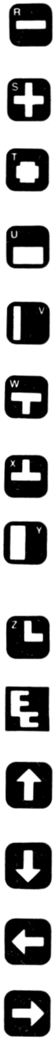
FB

We zien hoe het getal 11111011 gesplitst wordt in 1111 en 1011. Ieder gedeelte vatten we nu op als een 'gewoon' (positief) binair getal waarvan we eenvoudig de decimale en dus ook de hexadecimale waarde kunnen bepalen. Op deze wijze zien we hoe de aanduiding 11111011 uiteindelijk in FB wordt omgezet.
Het hexadecimale getal $FB komt overeen met het decimale getal 255 (= $15 \times 16^1 + 11$) en dit is ten slotte het getal dat we in de POKE-instructie opnemen als we het programma in het geheugen 'POKEn'.

Hiermee besluiten we onze 'korte' introductie. Als u meer van het programmeren in machine-instructies wilt weten raden we het bekende boek '6502 Programming' van Rodnay Zaks aan.

Appendix 1. Symbolen, schrifttekens en ASCII-codes

NORMAAL: Kleurregister 1 (groen) INVERS: Kleurregister 3 (rose)									
ASCII-code		Mode 0	Mode 1 en 2		ASCII-code		Mode 0	Mode 1 en 2	
decimaal	hexadecimaal	standaard karakterset	standaard karakterset	alternatieve karakterset	decimaal	hexadecimaal	standaard karakterset	standaard karakterset	alternatieve karakterset
0	0		Space		9	9		)	
1	1		!		10	A		*	
2	2		"		11	B		+	
3	3		#		12	C		,	
4	4		$		13	D		-	
5	5		%		14	E		.	
6	6		&		15	F		/	
7	7		'		16	10		0	
8	8		(		17	11		1	

NORMAAL: Kleurregister 1 (groen) INVERS: Kleurregister 3 (rose)											
ASCII-code d.	h.	Mode 0 s.k.	Mode 1 en 2 s.k.	a.k.	ASCII-code d.	h.	Mode 0 s.k.	Mode 1 en 2 s.k.	a.k.
18	12		2		33	21	!	!	
19	13		3		34	22	"	"	
20	14		4		35	23	#	#	
21	15		5		36	24	$	$	
22	16		6		37	25	%	%	
23	17		7		38	26	&	&	
24	18		8		39	27	'	'	
25	19		9		40	28	(	(	
26	1A		:		41	29	)	)	
27	1B		;		42	2A	*	*	
28	1C		<		43	2B	+	+	
29	1D		=		44	2C	,	,	
30	1E		>		45	2D	-	-	
31	1F		?		46	2E	.	.	
32	20	Space	Space		47	2F	/	/	

NORMAAL: Kleurregister 0 (oranje) (vanaf 32) INVERS: Kleurregister 2 (blauw) (vanaf 32)									
ASCII-code		Mode 0	Mode 1 en 2		ASCII-code		Mode 0	Mode 1 en 2	
d.	h.	s.k.	s.k.	a.k.	d.	h.	s.k.	s.k.	a.k.
48	30	0	0		63	3F	?	?	
49	31	1	1		64	40	@	@	
50	32	2	2		65	41	A	A	a
51	33	3	3		66	42	B	B	b
52	34	4	4		67	43	C	C	c
53	35	5	5		68	44	D	D	d
54	36	6	6		69	45	E	E	e
55	37	7	7		70	46	F	F	f
56	38	8	8		71	47	G	G	g
57	39	9	9		72	48	H	H	h
58	3A	:	:		73	49	I	I	i
59	3B	;	;		74	4A	J	J	j
60	3C	<	<		75	4B	K	K	k
61	3D	=	=		76	4C	L	L	l
62	3E	>	>		77	4D	M	M	m

NORMAAL: Kleurregister 0 (oranje) INVERS: Kleurregister 2 (blauw)									
ASCII-code		Mode 0	Mode 1 en 2		ASCII-code		Mode 0	Mode 1 en 2	
d.	h.	s.k.	s.k.	a.k.	d.	h.	s.k.	s.k.	a.k.
78	4E	N	N	n	93	5D	[	[	
79	4F	O	O	o	94	5E	^	^	
80	50	P	P	p	95	5F	-	-	
81	51	Q	Q	q	96	60		@	
82	52	R	R	r	97	61	a	A	a
83	53	S	S	s	98	62	b	B	b
84	54	T	T	t	99	63	c	C	c
85	55	U	U	u	100	64	d	D	d
86	56	V	V	v	101	65	e	E	e
87	57	W	W	w	102	66	f	F	f
88	58	X	X	x	103	67	g	G	g
89	59	Y	Y	y	104	68	h	H	h
90	5A	Z	Z	z	105	69	i	I	i
91	5B	[	[		106	6A	j	J	j
92	5C	\	\			107	6B	k	K	k

NORMAAL: Kleurregister 1 (groen) (vanaf 96) INVERS: Kleurregister 3 (rose) (vanaf 96)									
ASCII-code		Mode 0	Mode 1 en 2		ASCII-code		Mode 0	Mode 1 en 2	
d.	h.	s.k.	s.k.	a.k.	d.	h.	s.k.	s.k.	a.k.
108	6C	l	L	l	120	78	x	X	x
109	6D	m	M	m	121	79	y	Y	y
110	6E	n	N	n	122	7A	z	Z	z
111	6F	o	O	o	123	7B		[	
112	70	p	P	p	124	7C	|		|
113	71	q	Q	q	125	7D		]	
114	72	r	R	r	126	7E		^	
115	73	s	S	s	127	7F		-	
116	74	t	T	t	253	FD			
117	75	u	U	u	254	FE			
118	76	v	V	v	255	FF			
119	77	w	W	w					

Appendix 2. BASIC-functies

In deze appendix worden alle functies die ons Atari-BASIC kent nogmaals op een rijtje gezet. Zoals nu wel bekend is, bestaat elke functie uit een naam, één of meer argumenten en natuurlijk een resultaat.
In tegenstelling tot de commando's en instructies zijn de functienamen niet af te korten. In de onderstaande lijst van functies gebruiken we een aantal begrippen die eerst dienen te worden verklaard.

<getal>	elk mogelijk getal, bijv. 1.35E+13.
<string>	serie schrifttekens, bijv. "ROB".
<getal-expressie>	een berekening met getallen, variabelen of functies. Voorbeelden van een geldige <getal-expressie>:

```
4
A
A+4-SIN(X)
A*4+SIN(LEN("ATARI"))
```

<string-expressie>	een bewerking met strings.
<string-variabele>	een naam van een variabele waar een string in mag worden opgeslagen.

Verder gebruiken we steeds een zelfde syntax om aan te geven hoe de functie moet worden gebruikt:

naam (argumentenlijst) = resultaat

Nu zullen we een lijst geven van alle functies.

Wiskundige functies

ABS(<getal-expressie>)=<getal>
Levert de absolute waarde van het argument.

CLOG(<getal>-expressie>)=getal
Levert de logaritme van het argument met tien als grondtal. CLOG(0) zal een foutmelding geven en CLOG(1) is altijd 0.

EXP(<getal-expressie>)=<getal>
Levert de e-macht van het argument.

INT(<getal-expressie>)=<getal>
Levert de naar beneden afgeronde waarde van het argument. Bijvoorbeeld: INT(3.9)=3 en INT(-14.2)=-15.

LOG(<getal-expressie>)=<getal>
Levert de natuurlijke logaritme van het argument.

RND(<getal-expressie>)=<getal>
Levert een willekeurig getal tussen 0 en 1 (maar nooit het getal 1 zelf). Het argument heeft geen enkele invloed op het verloop van de functie en wordt daarom dummy-waarde genoemd. Een eenvoudige manier om een 'random-waarde' tussen a en b te krijgen is de volgende: X=INT(b*RND (0)+a), waarbij X het gewenste getal wordt.

SGN(<getal-expressie>)=<getal>
Deze functie wordt ook wel de SiGN-functie genoemd. Wanneer het argument namelijk negatief is zal <getal> de waarde -1 aannemen, bij een positief argument de waarde 1 en anders de waarde 0 (dus alleen als het argument ook 0 is).

SQR(<getal-expressie>)=<getal>
Levert de wortel van het argument. Bij een negatief argument zal er een foutmelding op het scherm verschijnen.

ATN(<getal-expressie>)=<getal>
Levert de arctangens van het argument.

COS(<getal-expressie>)=<getal>
Levert de cosinus van het argument. Het argument is in radialen, tenzij het met DEG is gewijzigd in graden. Het resultaat is altijd een getal tussen -1 en 1.

SIN(<getal-expressie>)=<getal>
Levert de sinus van het argument. Het argument is in radialen, tenzij het met DEG is gewijzigd is graden. Het resultaat is altijd een getal tussen -1 en 1.

DEG (geen argument en geen resultaat!)
Heeft tot gevolg dat de goniometrische functies op basis van graden (Engels: DEGrees) worden berekend.

RAD (geen argument en geen resultaat!)
Heeft tot gevolg dat de goniometrische functies op basis van radialen (Engels: RADians) worden berekend.

Speciale functies

ADR(<string-variabele>)=<getal>
Levert het decimale geheugenadres van de bewuste string.

FRE(<getal-expressie>)=<getal>
Levert het aantal ongebruikte bytes van het geheugen. Het argument is een dummy-argument.

PEEK(<getal-expressie>)=<getal>
Levert de waarde van het geheugenadres, aangeduid door het argument. <getal> krijgt hierbij een waarde van minimaal 0 tot maximaal 255.

String-functies

ASC(<string-expressie>)=<getal>
Levert de ASCII-waarde van het eerste karakter van het argument.

CHR$(<getal-expressie>)=<string>
Levert het karakter op dat behoort bij de ASCII-waarde die het argument aangeeft. Het resultaat is dus altijd een string met lengte 1. Het argument moet een waarde opleveren tussen 0 en 255.

LEN(<string-expressie>)=<getal>
Levert de lengte van de door het argument 'berekende' string. Het resultaat is dus altijd een geheel getal.

STR$(<getal-expressie>)=<string>
Levert een string die de leesbare vorm van de waarde van het argument bevat, zoals die met behulp van PRINT zou worden afgedrukt.

VAL(<string-expressie>)=<getal>
Levert de waarde van het getal dat wordt voorgesteld door de inhoud van het argument. Het argument moet dezelfde vorm hebben als een getal dat u in een programma kunt gebruiken (bijvoorbeeld: 1.7E-8 of 3.45).

Appendix 3. BASIC-commando's en -instructies

In deze appendix geven we een overzicht van alle BASIC-commando's en -instructies die de Atari kent. Steeds als een nieuwe instructie of een nieuw commando besproken gaat worden, geven we daarvan eerst een syntax-beschrijving. De syntax-beschrijving wordt dan nog gevolgd door één of meer mogelijke afkortingen die van toepassing zijn op de instructie of op het commando. In de syntax-beschrijvingen staan bepaalde symbolen en begrippen die een bepaalde betekenis hebben. We geven daarom hieronder eerst een korte uitleg van die gebruikte symbolen en begrippen.

< ... >	gegevens tussen deze haken moeten door de gebruiker zelf worden ingevuld.
[...]	gegevens tussen deze vierkante haken zijn optioneel, dat wil zeggen, ze mogen worden weggelaten.
device	hier wordt een stuk hardware mee bedoeld:

'K:'	is toetsenbord
'P:'	is printer
'C:'	is cassetterecorder
'D1:', 'D2:', 'D3:' en 'D4:'	zijn de disk drives
'E:'	is het scherm
'S:'	is de monitor
'R:'	is de RS-232-interface

filenaam	is een naam voor een file. Deze bestaat altijd uit drie delen: de naam (maximaal 8 karakters), een punt (.) en een extensie (zegt iets over het soort file).
variabele	is een variabele die niet is bepaald.
expressie	is een uitdrukking die ofwel een string, ofwel een getal oplevert. Uit de tekst blijkt dan wat daadwerkelijk het resultaat moet zijn.

BYE (B.)

Verlaat BASIC. De computer komt nu in de SELF TEST-mode. De gebruiker kan nu kiezen tussen drie tests: geheugentest, geluid-beeldschermtest en toetsenbordtest. Met de RESET-toets gaat men terug naar BASIC. Het geheugen van de computer bevat geen programma's meer.

CLOAD (CLOA.)
Maakt dat de computer het eerstvolgende programma van cassette zal lezen. Eenmaal op de RETURN-toets gedrukt laat de computer een toon horen. Dan dient de cassetterecorder op PLAY te worden gezet. Als er dan nóg een keer op de RETURN-toets wordt gedrukt, begint de computer het programma van de cassette te lezen.

CLOSE #<expressie> (CL.)
Deze instructie maakt een OPEN-instructie ongedaan. De <expressie> is aan dezelfde eisen onderworpen als bij de OPEN-instructie. De CLOSE-instructie mag ook worden gebruikt voor een file die niet is geOPENd. Er gebeurt dan niets. Zie verder bij OPEN.

CONT (CON.)
Voor het voortzetten van een programma. Wanneer een programma nog niet gelopen heeft, zal als gevolg van het commando CONT het programma bij de eerste regel beginnen. Is een programma door een foutmelding onderbroken of is de BREAK-toets ingedrukt, dan zal de regel waarin de fout zit of de regel waarin het programma is onderbroken niet nogmaals worden uitgevoerd, maar zal de computer het programma uitvoeren vanaf de daaropvolgende regel. Wanneer de BREAK-toets is gebruikt tijdens een INPUT-instructie, zal na het CONT-commando het programma na de INPUT-instructie verder gaan. Aan de INPUT-variabele wordt dan de waarde 0 of de lege string toegekend.

CSAVE (CS.)
Maakt het mogelijk om programma's vanuit het RAM-geheugen op cassette te zetten. Net als bij CLOAD zal na het intoetsen van het commando een toon klinken. Dan moet de cassetterecorder op opnemen worden gezet en zal na het nog eens intoetsen van de RETURN-toets het programma op cassette worden gezet.

DATA <datalijst>, READ <varlijst>, RESTORE [<regelnummer>] (D.), (REA.), (RES.)
Met deze instructies kan men een lijst van vaste waarden opnemen in het programma (met DATA), die dan één voor één kunnen worden opgehaald (met READ). Wanneer BASIC een regel tegenkomt die met DATA begint, slaat hij deze regel in eerste instantie over. Komt BASIC echter een READ-instructie tegen, dan zoekt hij de DATA-regel op, waar hij het laatst uit heeft gelezen en gaat daar verder met het inlezen van de DATA-lijst. De volgorde van het lezen van de DATA kan door de RESTORE-instructie worden veranderd. Na een RESTORE <regelnummer> wordt verder gelezen in de eerste DATA-regel na het gespecificeerd regelnummer. Wordt bij een RESTORE-instructie geen regelnummer meegegeven dan gaat de DATA-pointer (letterlijk: DATA-aanwijzer) naar de eerste DATA-regel van het programma. Dit is ook het geval nadat het RUN-commando is gegeven.
Ook string-variabelen mogen in de <varlijst> (lijst van variabelen) van READ voorkomen. De gespecificeerde DATA worden dan automatisch opgevat als string-variabelen. Let wel, er mogen geen aanhalingstekens omheen staan.

DOS (DO.)
Het DOS-commando wordt gebruikt om van BASIC over te gaan naar het Disk Operating System. Is men niet in het bezit van een floppy disk, dan zal de computer in plaats van DOS, de zelf-test-mode kiezen. In beide gevallen moet men de RESET-toets gebruiken om terug te keren naar BASIC. Alle programma's zullen dan uit het RAM-geheugen verdwenen zijn.

END (geen afkorting)
Beëindigt het lopen van een programma. De END-instructie hoeft niet aan het einde van het programma te staan. Met het CONT-commando kan dan alsnog de loop van het programma worden voortgezet. Als gevolg van END worden alle files gesloten en ook de geluidsgeneratoren worden uitgezet als zij aanstaan. De STOP-instructie doet dit niet (zie STOP).

ENTER <device[>[;<filenaam>] (E.)
Met dit BASIC-commando kan men de programma's die men met LIST op cassette heeft gezet (bijvoorbeeld LIST "C:PROGRAMMA"), weer in het geheugen laden (bijvoorbeeld ENTER "C:PROGRAMMA").

FOR...NEXT (F.) (N.)
De volledige syntax luidt:

```
FOR <variabele>=n TO m [STEP k]
...
NEXT <variabele>
```

Tussen FOR en NEXT kunnen willekeurig veel instructies worden gezet. De symbolen n, m en k stellen hier numerieke uitdrukkingen voor. Alle instructies tussen FOR en NEXT worden herhaald uitgevoerd. Net zolang totdat de teller (=<variabele>) de waarde van m overschrijdt. De beginwaarde van de teller is n en de teller wordt, iedere keer dat alle instructies zijn uitgevoerd, met k verhoogd. Als men STEP k weglaat, dan wordt de teller steeds automatisch met 1 verhoogd.

GET#<expressie>,<variabele> (GE.)
Deze instructie haalt via de gespecificeerde data-stroom één byte binnen en kent die toe aan de gespecificeerde <variabele>. De PUT-instructie is de omgekeerde bewerking hiervan.

GOSUB <variabele>
Deze instructie maakt het mogelijk om naar een subroutine te springen. <regelnummer> is het regelnummer van de eerste regel van deze subroutine. Een RETURN moet aan het eind van de subroutine voorkomen en zorgt er voor dat wordt teruggekeerd naar de eerstvolgende instructie na GOSUB.

GOTO <regelnummer> (G.)
De programma-uitvoering wordt vervolgd bij het opgegeven <regelnummer>.

IF...THEN geen afkortingen)
De volledige syntax luidt:

```
IF <expressie> THEN <regelnummer>
IF <expressie> THEN <instructies>
```

Met expressie wordt bedoeld: voorwaarde. Als aan de voorwaarde is voldaan (dat wil zeggen: de waarde die de expressie oplevert is ongelijk aan nul), dan wordt er gesprongen naar <regelnummer> ofwel de instructies worden uitgevoerd. Wordt echter niet aan de voorwaarde voldaan (dat wil zeggen: de expressie levert nul op), dan wordt alles wat na THEN komt overgeslagen.

INPUT[#<device>]<variabele>[,<variabele> ...] (I.)
Deze instructie verwacht respons van het toetsenbord. De <variabele> mag ook een string-variabele zijn. Wanneer geen <device> is gespecificeerd, zal de computer automatisch aannemen dat de <device> "E:" wordt bedoeld (screen editor). Elementen van matrices (twee-dimensionale arrays) mogen niet als variabele worden opgegeven.

LIST ["<device>:<filenaam>",][<regelnummer>[,<regelnummer>]] (L.)
Alle programmaregels van het eerste regelnummer tot het tweede regelnummer (tweede regelnummer inbegrepen) worden 'geLIST'. De opgegeven regelnummers hoeven niet te bestaan. Wanneer geen device wordt meegegeven verschijnt de listing op het beeldscherm. Wordt slechts één regelnummer gespecificeerd, dan wordt deze alleen bedoeld. De LIST-instructie zonder regelnummer betekent dat het gehele programma wordt bedoeld.

LOAD "<device>;<filenaam>"
Dit commando lijkt erg veel op het CLOAD-commando. Het enige verschil is dat nu een <filenaam> moet worden opgegeven, evenals een <device>.

LPRINT [<expressie>[(, of ;)<expressie>]] (LP.)
Deze instructie lijkt erg veel op de 'gewone' PRINT-instructie. Het enige verschil is dat er nu op de printer wordt geprint (de L die voor PRINT staat komt van 'lijnprinter'). Er hoeft geen OPEN-instructie te worden gebruikt. De computer gebruikt zelf het datastroomnummer 7 hiervoor.

NEW (geen afkorting)
Dit commando verwijdert alle programma's uit het RAM-geheugen. Het is aan te bevelen dit commando niet in een programma te gebruiken, tenzij daar een zeer speciale reden voor is.

NEXT (N.)
Zie FOR...NEXT.

NOTE #<expressie>,<variabele1>,<variabele2> (NO.)
Deze instructie kan alleen worden gebruikt als ook een disk drive is aangesloten. De expressie moet een waarde opleveren die afgerond een geheel getal van minimaal 1 tot maximaal 7 oplevert. Aan de eerste variabele wordt dan het sectornummer toegekend en aan de tweede variabele het nummer van de byte waar de schrijf- en leeskop van de disk drive zich bevindt.

ON <expressie> GOSUB <regelnummerlijst> ... RETURN (GOS.) (RET.)
De expressie wordt eerst door BASIC afgerond op een geheel getal. Vervolgens wordt er gesprongen naar dat regelnummer in de <regelnummerlijst> dat overeenkomt met het gehele getal. (Bijvoorbeeld de <expressie> levert

3.2 op; dit wordt afgerond naar het gehele getal 3; derhalve wordt naar het derde regelnummer gesprongen.)
Dit regelnummer is het begin van een subroutine. In deze subroutine moet de instructie RETURN voorkomen, waarop het programma zal terugkeren naar de eerstvolgende instructie na de ON...GOSUB-instructie.

ON <expressie> GOTO <regelnummerlijst>
Deze instructie lijkt erg veel op de vorige. Het enige verschil is dat bij deze instructie niet naar een subroutine wordt gesprongen, maar dat het programma eenvoudig wordt voortgezet op een regelnummer uit de <regelnummerlijst>.

OPEN #<expressie>,n1,n2,"<device>:<filenaam>" (O.)
Voordat men een bepaald device (randapparaat, bijv. disk drive) kan bereiken, moet men een OPEN-instructie gebruiken. De <expressie> moet hier een getal opleveren dat afgerond tussen 1 en 7 ligt. De computer weet dan dat met deze waarde (na het karakter '#') steeds de <device> wordt bedoeld, die samen met het getal is gespecificeerd. Wij zullen steeds spreken van een datastroom. Op de plaats n1 komt een getal te staan. Dit getal geeft informatie over de soort van operatie. Als n1=4 dan wordt een leesoperatie bedoeld, met n1=8 een schrijfoperatie en met n1=12 een lees- of schrijfoperatie. Met n1=6 wordt aangeduid dat in de disk directory (en overzicht van de programma's op de diskette) wordt gezocht; er wordt dan naar <filenaam> gezocht. De code n1=9 heeft ten slotte een tweeledig doel. De code geeft ten eerste de end-of-file aan. Is echter de <device> "E:" gespecificeerd dan mag de volgende hele regel van het scherm worden ingevoerd (vanuit een programma), zonder dat de computer op een RETURN wacht.
Het getal n2 geeft ook een code aan, maar de betekenis daarvan is voor elk <device> weer anders. De juiste control-codes zijn terug te vinden in de daarvoor bedoelde handleidingen.

POINT #<expressie>,<variabele1>,<variabele2> (P.)
Deze instructie doet als het ware het omgekeerde van de NOTE-instructie. In plaats van het uitlezen van de positie van de schrijf- en leeskop, zorgt deze instructie dat de schrijf- en leeskop naar een bepaalde positie wordt toegestuurd. De eerste variabele moet dan het nummer bevatten van de sector en de tweede variabele het nummer van de byte van de nieuwe positie. In de DOS-handleiding staat meer gedetailleerde informatie over het gebruik van de NOTE- en de POINT-instructie.

POP (geen afkorting)
De POP-instructie heeft maar één doel en wel het ontlopen van een RETURN-instructie. Wanneer namelijk onder zekere omstandigheden uit een subroutine wordt gesprongen zonder dat daarvoor de RETURN-instructie is gebruikt, mag in plaats daarvan de POP-instructie worden gebruikt.

PRINT [#<exp>,] (, of ;) [<exp>(, of ;)[<exp>...]...] (PR of ?)
Er komen drie expressies voor in deze instructie (<exp>). De eerste moet afgerond een getal tussen 1 en 7 opleveren. De tweede expressie (en elke volgende) mag elke expressie zijn mits het correct BASIC is. Deze

expressie wordt dan geprint. De derde expressie staat voor een willekeurige hoeveelheid andere expressies. Deze moeten steeds worden gescheiden van de vorige door middel van een komma (,) of een puntkomma (;). Wanneer voor de komma wordt gekozen zal de expressie die er op volgt worden voorafgegaan door tien spaties. Wanneer echter de puntkomma werd meegegeven, dan komt de expressie pal achter de vorige te staan.

PUT #<expressie>,<expressie> (PU.)
De eerste expressie moet een getal opleveren dat afgerond een geheel getal van 1 tot 7 oplevert. De tweede expressie moet een byte opleveren (getal 0-255). Deze enkele byte wordt weggeschreven via de gespecificeerde datastroom uit de eerste expressie.

READ <varlijst> (REA.)
Zie DATA.

REM (R. of .) (punt, gevolgd door spatie)
Stelt de programmeur in de gelegenheid om opmerkingen in de programma-tekst te plaatsen. Bij uitvoering van het programma door de computer worden alle REM-instructies overgeslagen met alles wat er achter staat.

RESTORE (RES.)
Zie DATA.

RETURN (RET.)
Zie GOSUB.

RUN ["device:filenaam"] (RU.)
Start de uitvoering van het aangegeven programma. Als er geen programma wordt aangeduid, dan wordt het programma uitgevoerd dat zich in het RAM-geheugen van de computer bevindt.

SAVE "<device>:<filenaam>" (S.)
Dit commando lijkt erg veel op het CSAVE-commando. Het enige verschil is dat nu een <device> moet worden meegegeven en bovendien een <file-naam>.

STOP (STO.)
Stopt de uitvoering van het programma. BASIC geeft dan de melding STOPPED AT LINE ... waarbij op de plaats van de puntjes het regelnummer staat waar de uitvoering is gestopt. De STOP-instructie zet geen geluidsgeneratoren af en sluit ook geen files, zoals de END-instructie wel doet.

TO (geen afkorting)
Zie FOR...NEXT.

TRAP <regelnummer> (T.)
De TRAP-instructie wordt gebruikt om naar een bepaalde regel te springen wanneer er een fout in het programma optreedt. Normaal gesproken zou BASIC het uitvoeren van het programma staken. TRAP reageert op elke fout die kan voorkomen als het programma eenmaal loopt, mits de TRAP-instructie al is uitgevoerd. PEEK(195) geeft het error-nummer

en 256 *PEEK(187)+PEEK(186) geeft de programmaregel waarin de fout is opgetreden. Een TRAP-instructie kan ongedaan worden gemaakt door een TRAP-waarde tussen 32767 en 65536 te geven, maar wordt automatisch ongedaan gemaakt wanneer een fout is 'getrapt' (letterlijk: gestrikt).

XIO nr,#<exp>,<exp1>,<exp2>,"<device>:<filenaam>" (X.)

Deze instructie is zonder twijfel de ingewikkeldste die het Atari-BASIC kent. Eén voorbeeld hebben we al gezien in een van de hoofdstukken over de grafische mogelijkheden en wel met de toepassing om een gebied op te vullen op het grafische scherm. De volgende tabel geeft een overzicht van wat de XIO-instructie allemaal kan. Nr in de syntax van de XIO-instructie staat voor een getal dat de instructie aangeeft.

Nr	Operatie	Voorbeeld
3	OPEN	gelijk aan BASIC OPEN
5	GET RECORD	gelijk aan BASIC INPUT
7	GET karakters	gelijk aan BASIC GET
9	PUT RECORD	gelijk aan BASIC PRINT
11	PUT karakters	gelijk aan BASIC PUT
12	CLOSE	gelijk aan BASIC CLOSE
17	DRAW lijn	gelijk aan BASIC DRAWTO
18	FILL	zie boven
32	RENAME	XIO 32,#1,0,0,"D:TEMP.TXT"
33	DELETE	XIO 33,#1,0,0,"D:TEMP.BAS"
35	LOCK file	XIO 35,#1,0,0,"D:TEMP.BAS"
36	UNLOCK file	XIO 36,#1,0,0,"D:TEMP.BAS"
37	POINT	gelijk aan BASIC POINT
38	NOTE	gelijk aan BASIC NOTE
254	FORMAT	XIO 254,#1,0,0,"D2"

<exp> is het nummer van de datastroom. De expressie moet een getal opleveren dat afgerond een getal van 1 tot maximaal 7 is.
<exp1> en <exp2> zijn twee bytes, die voor elke <device> weer een andere functie hebben. Meestal worden ze niet gebruikt en zijn gewoon nul.

Appendix 4. MODE's en kleuren

Legende:
A kleurregister
B selectie van de kleur
C functie van het kleurregister

MODE 0. Alle tekstvensters

A	B	C
0	gebruik	niet gebruikt
1	SETCOLOR-instructie	lichtsterkte van tekens, de kleur is dezelfde als die van de achtergrond
2		achtergrond
3		niet gebruikt
4		rand van het scherm

MODE 1 en 2. Tekstmodes

A	B	C
0	gebeurt	karakters
1	door teken-	karakters uit alternatieve karakterset
2	conversie	inverse karakters
3		inverse karakters uit alternatieve karakterset
4		achtergrond en rand van het scherm

MODE 3, 5 en 7. Grafische modes met vier kleuren

A	B	C
	gebruik COLOR-opdracht	
0	1	grafische punt (pixel)
1	2	grafische punt (pixel)
2	3	grafische punt (pixel)
3	-	niet gebruikt
4	0	grafische punt (pixel) en achtergrond en rand van het scherm

MODE 4 en 6. Grafische modes met twee kleuren

A	B gebruik COLOR-opdracht	C
0	1	grafische punt (pixel)
1	-	niet gebruikt
2	-	niet gebruikt
3	-	niet gebruikt
4	0	grafische punt (pixel) en achtergrond en rand van het scherm

MODE 8. Grafische mode met één kleur en twee lichtsterktes

A	B gebruik COLOR-opdracht	C
0	1	niet gebruikt
1	-	lichtsterkte van grafische punt, de kleur is dezelfde als die van de achtergrond
2	0	grafische punt en achtergrond
3	-	niet gebruikt
4	-	niet gebruikt

Appendix 5. MODE's, geheugen en schermafmetingen

A	B	C	D	E	F	G
0	tekst	40	-	24	2	993
1	tekst	20	20	24	5	513
2	tekst	20	10	12	5	216
3	grafisch	40	20	24	4	273
4	grafisch	80	40	48	2	537
5	grafisch	80	40	48	4	1017
6	grafisch	160	80	96	4	2025
7	grafisch	160	80	96	4	3945
8	grafisch	320	160	192	1.5	7900

Legende:

A grafische mode
B soort mode
C horizontaal, aantal kolommen
D verticaal, aantal rijen. Hierbij is het scherm verdeeld in een tekstdeel en in een grafisch deel.
E verticaal, aantal rijen. Het gehele scherm is grafisch (mode +16).
F aantal kleuren
G benodigd geheugen (bytes)

Voorbeeld: In mode 4 bestaat het scherm uit een klein tekstgedeelte en een grafisch gedeelte. Het grafische deel is 80 pixels breed (zie C) en 40 pixels hoog (zie D). In mode 20 (4+16) is het grafische deel 48 pixels hoog (zie E). Er is dan geen tekstgedeelte meer.

Appendix 6. Nuttige geheugenadressen

Naam	**Adresplaats**	**Betekenis**
RTCLOK	18, 19, 20	Klok, teller, telt automatisch elke 1/50 seconde 1 op (LSB, NSB, MSB).
SOUNDNR	65	Ruis (0=stil).
	77	Attract mode (automatisch wisselen van beeldkleur, wordt door toetsindruk gereset (128=attract mode).
LMARGIN	82	Positie linkerkantlijn (normaal 2).
RMARGIN	83	Positie rechterkantlijn (normaal 39).
ROWCRS	84	Huidige rij (regel) van de cursor (grafisch venster).
COLCRS	85, 86	Huidige kolom van de cursor (grafisch venster).
OLDROW	90	Vorige rij (regel) van de cursor (grafisch venster).
OLDCOL	91, 92	Vorige kolom van de cursor (grafisch venster).
	93	Data op de plaats van de cursor (grafisch venster, behalve in mode 0).
NEWROW	96	Verticale coördinaat van eindpunt van DRAWTO.
NEWCOL	97, 98	Horizontale coördinaat van eindpunt van DRAWTO.
RAMTOP	106	Momentele top van het geheugen, in aantallen pagina's.
STOPLN	186, 187	Regelnummer waar een STOP op TRAP plaatsvond.
ERRSAV	195	Foutnummer.
PTABW	201	Tabulatorbreedte (normaal 10).
FRO	212, 213	Laag en hoog orde-byte voor BASIC na RETURN van USR-functie.
RADFLG	251	Selectie radialen/graden (0=radialen, 6=graden).
LPENH	564	Lichtpen horizontale waarde.
LPENV	565	Lichtpen verticale waarde.
	599	Video-besturing (0=scherm uit, 34=scherm aan).

Naam	Adresplaats	Betekenis
TXTROW	656	Rij (regel) van de cursor (tekstvenster).
TXTCOL	657, 658	Kolom van de cursor (tekstvenster).
	694	Cursor-toestand (0=normaal, 128= inverse video).
COLOR0	708	Kleurregister 0.
COLOR1	709	Kleurregister 1.
COLOR2	710	Kleurregister 2.
COLOR3	711	Kleurregister 3.
COLOR4	712	Kleurregister 4.
CRSINH	752	Cursor-toestand (0=cursor aan, 1=cursor uit).
CHACT	755	Karaktersoort register (4=op de kop, 2=normaal, 1=cursor uit).
CHBAS	756	Karakterset adres (normaal 224), (244= gewone karakterset met hoofdletters, 266=alternatieve karakterset met kleine letters).
ATACHR	763	Laatste ASCII-teken.
CH	764	Laatst ingetoetste toets; interne code (255=geen karakter).
FILDAT	765	Vulkleur voor grafische XIO-instructie.
DSPFLG	766	Display control karakters (0=normaal, 1=display).
SSFLAG	767	Uitvoertoestand (0=aan; normaal, 1=onderbroken), gezet door control-1.
	842	Invoer (12=normaal van toetsenbord, 13= van scherm).
CONSOL	53279	Besturingstoetsen (bit 2=OPTION; bit 1= SELECT; bit 3=START), POKE 53279,0 alvorens te lezen (0=toets ingedrukt).
PACTL	54018	Poort A controleregister (voor recorder: 52=aan, 60=uit).
SKCTL	53775	Seriepoort controleregister (bit 2=0 als de laatste toets nog steeds is ingedrukt).

Appendix 7. Foutboodschappen

In deze appendix wordt een overzicht gegeven van de foutboodschappen die de Atari toont als er een fout wordt geconstateerd. Voorbeeld:

```
10 LET A=0
20 PRINT 20/A
30 END
```

Als we het RUN-commando geven verschijnt:

```
ERROR- 11 AT LINE 20
```

Als we willen weten wat deze foutmelding inhoudt, dan moeten we weten wat de computer bedoelt met het getal 11. Verder vertelt de computer ons dat de fout in regel 20 was geconstateerd. Soms is het niet nodig om de fout op te zoeken, omdat uit de regel al duidelijk blijkt welke fout is gemaakt. Vooral in het begin zal het nodig zijn om te weten welke fout is gemaakt. Dan is de hieronder uitgewerkte lijst met foutmeldingen onontbeerlijk.

Foutnummer	Betekenis
2	**Memory insufficient** Niet genoeg geheugen voor de nieuwe variabelenaam, de nieuwe array of de nieuwe string.
3	**Value error** Ofwel een negatief getal werd gevonden waar een positief getal werd verwacht, of een expressie neemt een niet toegestane waarde aan.
4	**Too many variables** In een programma mogen maximaal 128 namen voorkomen.
5	**String length error** Er wordt geprobeerd een string te maken die groter is dan zijn dimensie.

Foutnummer	Betekenis
6	**Out of data** Er wordt een READ-instructie gebruikt, terwijl de datapointer voorbij de laatste data-regel in het programma is.
7	**Number greater than 32767** Een waarde is geen positieve integer of is groter dan 32767.
8	**Input statement error** Er wordt geprobeerd om een niet numerieke waarde in te lezen in een numerieke variabele.
9	**Array or string DIM error** De DIM-grootte is groter dan 32767 of een gespecificeerd element valt buiten de dimensionering of de string of array is al gedimensioneerd of er wordt gebruik gemaakt van een niet gedimensioneerde string of array.
10	**Stack overflow** Letterlijk: de stack loopt over. Er zijn teveel geneste subroutines en/of te ingewikkelde expressies.
11	**Floating point overflow/underflow error** Poging om door 0 te delen of de waarde wordt groter dan 1E98 of kleiner dan 1E-99.
12	**Line not found** Er is een regel niet gevonden die was gespecificeerd na een GOSUB- of een GOTO- of een THEN-instructie.
13	**No matching FOR-statement** Van een FOR...NEXT-instructie wordt alleen maar de NEXT aangetroffen. Het aangegeven regelnummer is de regel waarin de losse NEXT is gevonden.
14	**Line too long error** De aangetroffen instructie is voor BASIC te lang of te ingewikkeld.
15	**GOSUB or FOR line deleted** De bij de GOSUB-instructie behorende RETURN of de bij de FOR-instructie behorende NEXT wordt niet meer aangetroffen bij terugkomst. De instructie is dus in de tussentijd verwijderd.
16	**RETURN error** Er wordt een RETURN aangetroffen zonder dat een GOSUB-instructie is uitgevoerd.
17	**Garbage error** Letterlijk betekent dit 'afval fout'. BASIC probeert nu

Foutnummer	Betekenis
	'slechte RAM-bits' te verwerken. Deze fout duidt op een hardware-probleem, maar kan ook het resultaat zijn van een verkeerd gebruik van de POKE-instructie.
18	**Invalid string character** Een string begint niet met een juist karakter of een string in een VAL-functie is geen numerieke string.
19	**LOAD program too long** Er is niet genoeg geheugen over om de LOAD-instructie af te maken.
20	**Device number error** Er wordt gebruik gemaakt van een device-waarde die groter is dan 7 of gelijk is aan 0.
21	**LOAD file error** Met de LOAD-instructie wordt geprobeerd een file in te lezen die daar niet voor geschikt is.
128	**BREAK abort** De gebruiker heeft de BREAK-toets ingedrukt tijdens een I/O-operatie.
129	**IOCB error** I/O-poort is al geOPENed.
130	**Nonexistent device error** Er wordt door de gebruiker een niet bestaand device gespecificeerd.
131	**IOCB write only** Er wordt een READ-instructie naar een 'schrijf-alleen'-device gestuurd (printer).
132	**Invalid command** Er wordt een commando naar een device gestuurd dat voor de betreffende device niet geldig is.
133	**Device or file not open** Er is geen OPEN-instructie gespecificeerd voor de betreffende device.
134	**Illegal device-number** Er wordt een device-nummer gebruikt dat niet bestaat.
135	**IOCB read-only error** Er wordt getracht te schrijven naar een device die alleen voor lezen is geschikt.

Foutnummer	Betekenis
136	**EOF** Het einde van een file is bereikt. Deze melding kan gedaan worden bij het gebruik van cassette-files.
137	**Truncated record** Er wordt getracht een record te lezen dat langer is dan 256 bytes.
138	**Device timeout** Het betreffende device geeft geen respons.
139	**Device NAK** Er wordt niet-leesbare informatie over de seriële poort gestuurd of er is een 'slechte' disk drive.
140	**Serial bus error** Er is een fout opgetreden bij het lezen van informatie over de bus.
141	**Cursor out of range error** Deze foutmelding betekent zoveel als: de cursor is buiten bereik. Deze foutmelding kan in bepaalde modes voorkomen.
142	**Serial bus data frame overrun** Er is bij het binnenkomen en inlezen van informatie over de bus een fout opgetreden.
143	**Serial bus data frame checksum error** Overgezonden informatie over de bus is niet correct (controle met checksum leverde de fout).
144	**Device done error** Een poging tot schrijven naar een beveiligde (write protected: er kan niet meer naar de schijf worden geschreven) diskette.
145	**Read after write compare error** Verkeerde beeldscherm-mode.
146	**Function not implemented** Er wordt getracht gebruik te maken van een functie die niet aanwezig is.
147	**Insufficient RAM** Er is niet genoeg RAM-geheugen om de grafische mode te openen.
160	**Drive number error** Er wordt een disk drive gespecificeerd die niet bestaat.

Foutnummer	Betekenis
161	**Too many open files** Er zijn teveel files tegelijkertijd open. Er is hierdoor geen plaats meer in de buffer.
162	**Disk full** Er wordt getracht iets op disk te zetten, terwijl deze vol is, of zojuist is volgeraakt. Het programma staat in het laatste geval niet volledig op diskette.
163	**Unrecoverable system data I/O error** Letterlijk: een niet te herstellen systeem input/output-fout.
164	**File number mismatch** De informatie op de schijf voor de computer is niet meer intact.
165	**Filename error** Er is een verkeerde/foute filenaam in het spel.
166	**POINT data length error** Record-lengte klopt niet.
167	**File locked** Een file op diskette is beveiligd en het zojuist gegeven commando kan daardoor niet worden uitgevoerd.
168	**Command invalid** Letterlijk: commando niet in orde.
169	**Directory full** Het programma past niet meer op de diskette omdat reeds 64 programma's op schijf staan (dit is het maximum).
170	**File not found** De computer kan een door de gebruiker aangeduide file niet vinden.
171	**POINT invalid** Record bestaat niet.

Appendix 8. Machine-instructies

Deze appendix geeft een samenvatting van de machine-instructies. Onderstaande tabel toont hoeveel woorden iedere instructie inneemt bij de verschillende adresseerwijzen. Bovendien zien we hoe iedere adresseerwijze kort wordt aangegeven. Zo geven we bijvoorbeeld IMMED met I aan.

I=IMMED	A=ABSOL.	Z=Z.PAGE	AC=ACCUM.	IM=IMPLIED
2	3	2	1	1

IX=(IND,X)	IY=(IND),Y	ZX=Z.PAGE,X	AX=ABS,X	AY=ABS,Y
2	2	2	3	3

R=RELAT.	IN=INDIR.	ZY=Z.PAG.Y
2	3	2

Bij de beschrijving van de machine-instructies worden verschillende letters gebruikt, namelijk:

A	voor ACCU
M	voor geheugenplaats
C	voor CARRY-bit
X	voor X-register
Y	voor Y-register
M_S	voor een geheugenplaats die door de stack-pointer wordt aangewezen
S	voor stack

De volgende bewerkingstekens worden gebruikt:

+	optellen
-	aftrekken
AND	bit voor bit vergelijken: alleen als beide bits 1 zijn is het resultaat 1
OR	bit voor bit vergelijken: het resultaat is 1 als tenminste één van de bits 1 is
XOR	bit voor bit vergelijken: alleen als een van beide bits 1 is zal het resultaat 1 zijn

Voorbeelden:

A	= 10010110
M	= 01011010
A AND M	= 00010010
A OR M	= 11011110
A XOR M	= 11001100

Bij iedere instructie wordt aangegeven hoe het processor statusregister (P) gewijzigd kan worden. Hierbij worden de volgende tekens gebruikt:

X kan gewijzigd worden
- blijft ongewijzigd
0 wordt 0
1 wordt 1

Bij de beschrijving van iedere instructie wordt een bijzondere notatie met pijltjes gebruikt die naar rechts wijzen, bijvoorbeeld:

M → A

Dit wil zeggen: inhoud M wordt in A geplaatst.

Instructie	**Omschrijving**	**Codes als functie van adresseerwijze**
ADC	A+M+C → A	I:69 A:6D Z:65 IX:61 IY:71 ZX:75 AX:7D AY:79 P=NZCIDV XXX--X
AND	A AND M → A	I=29 A:2D Z:25 IX=21 IY:31 ZX:35 AX=3D AY:3 P=NZCIDV XX----
ASL	alle bits schuiven een plaats naar links in bit 0 komt 0 en bit 7 komt in C	A:0E Z:06 AC:0A ZX:16 AX:1E P=NZCIDV XXX---
BCC	spring als C=0	R:90 P=ongewijzigd
BCS	spring als C=1	R:B0 P=ongewijzigd
BEQ	spring als Z=1	R:F0 P=ongewijzigd

Instructie	Omschrijving	Codes als functie van adresseerwijze
BMI	spring als N=1	R:30 P=ongewijzigd
BNE	spring als Z=0	R:D0 P=ongewijzigd
BPL	spring als N=0	R:10 P=ongewijzigd
BVC	spring als V=0	R:50 P=ongewijzigd
BIT	A AND M	A:2C Z:24 P= N wordt bit 7 V wordt bit 6 Z kan veranderen
CLC	0 → C	I:18 P=NZCIDV --0---
CLD	0 → D	I:D8 P=NZCIDV ----0-
CLI	0 → I	I:58 P=NZCIDV ---0--
CLV	0 → V	I:B8 P=NZCIDV -----0
CMP	A-M	IM:C9 A:CD Z:C5 IX:C1 IY:D1 ZX:D5 AX:DD AY:D9 P=NZCIDV XXX---
CPX	X-M	I:E0 A:EC Z:E4 P=NZCIDV XXX---
CPY	Y-M	I=C0 A:CC Z:C4 P=NZCIDV XXX---
DEC	M-1 → M	A:CE Z:C6 ZX:D6 AX:DE P=NZCIDV XX----

Instructie	Omschrijving	Codes als functie van adresseerwijze
DEX	X-1 → X	IM:CA P=NZCIDV XX----
DEY	Y-1 → Y	IM:88 P=NZCIDV XX----
EOR	A XOR M → A	I:49 A:4D Z:45 IX:41 IY:51 ZX:55 AX:5D AY:59 P=NZCIDV XX----
INC	M+1 → M	A:EE Z:E6 ZX:F6 AX:FE P=NZCIDV XX----
INX	X+1 → X	IM:E8 P=NZCIDV XX----
INY	Y+1 → Y	IM:C8 P=NZCIDV XX----
JMP	spring naar adres	A:4C IN:6C P=ongewijzigd
JSR	spring naar subroutine	A:20 P=ongewijzigd
LDA	M → A	I:A9 A:AD Z:A5 IX:A1 IY:B1 ZX:B5 AX:BD AY:B9 P=NZCIDV XX----
LDX	M → X	I:A2 A:AE Z:A6 AY:BE ZY:B6 P=NZCIDV XX----
LDY	M → Y	I:A0 A:AC Z:A4 ZX:B4 AX:BC P=NZCIDV XX----

Instructie	Omschrijving	Codes als functie van adresseerwijze
LSR	alle bits schuiven één plaats naar rechts bit 0 → C en bit 7 wordt 0	A:4E Z:46 AC:4A ZX:56 AX:5E P=NZCIDV 0XX---
NOP	geen effect (no operation)	IM:EA P=ongewijzigd
ORA	A OR M → A	I:09 A:0D Z:05 IX:01 IY:11 ZX:15 AX:1D AY:19 P=NZCIDV XX----
PHA	A → M_S en S-1 → S	IM:48 P=ongewijzigd
PHP	P → M_S en S-1 → S	IM:08 P=ongewijzigd
PLA	S+1 → S en M_S → A	IM:68 P=NZCIDV XX----
PLP	S+1 → S en M_S → P	IM:28 P=als M_S
ROL	alle bits één plaats naar links, bit 7 komt in C en C komt in bit 0	A:2E Z:26 AC:2A ZX:36 AX:3E P=NZCIDV XXX---
RTI	return vanuit interrupt	IM:40 P=wordt hersteld
RTS	return vanuit subroutine	IM:60 P=ongewijzigd
SBC	A-M-$\bar{C}$ → A $\bar{C}$ wil zeggen 0 als C=1 en andersom 1 als C=0	I:E9 A:ED Z:E5 IX:E1 IY:F1 ZX:F5 AX:FD AY:F9 P=NZCIDV XX --X Als bewerking een negatief resultaat oplevert wordt C=1
SEC	1 → C	IM:38 P=NZCIDV --1---

Instructie	Omschrijving	Codes als functie van adresseerwijze
SED	1 → D	IM:F8 P=NZCIDV - - - - 1 - Als het D-bit is 'geset' interpreteert de processor de getallen als decimale getallen (2 getallen per plaats). Ook bij de ADC-, SBC- en CMP-instructie wordt dan met decimalen gewerkt.
SEI	1 → I	IM:78 P=NZCIDV - - - 1 - -
STA	A → M	A:8D Z:85 IX:81 IY:91 ZX:95 AX:9D AY:99 P=ongewijzigd
STX	X → M	A:8E Z:86 ZY:96 P=ongewijzigd
STY	Y → M	A:8C Z:84 ZX:94 P=ongewijzigd
TAX	A → X	IM:AA P=NZCIDV XX- - - -
TAY	A → Y	IM:A8 P=NZCIDV XX- - - -
TSX	S → X	IM:BA P=NZCIDV XX- - - -
TXA	X → A	IM:8A P=NZCIDV XX- - - -
TXS	X → S	IM=9A P=ongewijzigd
TYA	Y → A	IM=98 P=NZCIDV XX- - - -

Programma's

Lineaire regressie

De regressie- en correlatieberekening houdt zich bezig met de vraag hoe men door een gegeven aantal punten zo goed mogelijk een rechte lijn kan trekken. Deze rechte lijn wordt door twee grootheden getypeerd, namelijk de hellingshoek M en het punt waar de lijn de Y-as doorsnijdt (B). Ieder punt wordt getypeerd door een X- en Y-coördinaat. Het programma vraagt allereerst hoeveel punten er worden ingevoerd.
Na het invoeren van de coördinaten worden dan de waarden M en B berekend. Ten slotte wordt de coördinatie-coëfficiënt R getoond. Als alle punten precies op een rechte lijn liggen zal de absolute waarde van R gelijk aan 1 zijn. Indien de punten niet op een rechte lijn liggen zal de absolute waarde van R minder dan 1 zijn. Kennelijk geeft de waarde van R aan, hoe goed de punten wel op de berekende rechte lijn liggen.

Programma

```
10 REM LINEAIRE REGRESSIE
20 DIM A$(1)
100 GOSUB 700:REM INITIALISATIE
110 GOSUB 200:REM INVOER
120 GOSUB 400:REM BEREKENING
130 GOSUB 600:REM UITVOER
140 PRINT
150 PRINT "1: NIEUWE BEREKENING"
151 PRINT "2: DOORGAAN MET OUDE WAARDEN"
152 PRINT "3: PROGRAMMA BEEINDIGEN"
155 PRINT:PRINT "KEUZE: (1/2/3)";:INPUT KEUZE
160 ON KEUZE GOTO 100,110,170
170 END
200 REM INVOER
210 TRAP 300
220 PRINT "VOER GETALLENPAAR IN: ";:INPUT X,Y
230 AANTAL=AANTAL+1
240 SOMX=SOMX+X:SOMY=SOMY+Y
250 KWADRAATX=KWADRAATX+X*X
260 KWADRAATY=KWADRAATY+Y*Y
```

```
270 XY=XY+X*Y
280 GOTO 220
300 IF PEEK(195)=8 THEN RETURN
310 PRINT "INVOER FOUT":STOP
400 REM BEREKENING
410 RICHTING=(SOMX*SOMY/AANTAL-XY)/(SOMX*SOMX/AANTAL
-KWADRAATX)
420 NULPUNT=SOMY/AANTAL-RICHTING*SOMX/AANTAL
430 T=AANTAL*XY-SOMX*SOMY
440 NN=SQR((AANTAL*KWADRAATX-SOMX*SOMX)*(AANTAL*
KWADRAATY-SOMY*SOMY))
450 CORRELATIE=T/NN
460 RETURN
600 REM UITVOER
610 PRINT "RICHTINGS-COEFFICIENT=";RICHTING
620 PRINT "SNIJPUNT MET Y-AS BIJ Y=";NULPUNT
630 PRINT "CORRELATIE-COEFFICIENT=";CORRELATIE
640 RETURN
700 REM INITIALISATIE
710 SOMX=0:SOMY=0
720 KWADRAATX=0:KWADRAATY=0
730 AANTAL=0:XY=0
740 RETURN
```

Voorbeeld:

```
VOER GETALLENPAAR IN: ?1,2
VOER GETALLENPAAR IN: ?2,4
VOER GETALLENPAAR IN: ?3,7
VOER GETALLENPAAR IN: ?4,6

RICHTINGS-COEFFICIENT=2.5
SNIJPUNT MET Y-AS BIJ Y=-0.66666667
CORRELATIE-COEFFICIENT=0.99332992678

1: NIEUWE BEREKENING
2: DOORGAAN MET OUDE WAARDEN
3: PROGRAMMA BEEINDIGEN

KEUZE (1,2,3)?3
```

Mastermind

Als er een spel de laatste jaren populair is geworden, dan is het wel 'Mastermind'. De charme van het spel ligt ongetwijfeld in zijn eenvoud. Oorspronkelijk wordt het spel met gekleurde pennetjes gespeeld. Bij de computer maken we gebruik van cijfers. De computer heeft vier cijfers onzichtbaar in vier vakjes gestopt en u dient die cijfers te raden door steeds vier cijfers op te noemen. De computer meldt dan hoeveel cijfers van het door u genoemde rijtje inderdaad in de vakjes liggen en bovendien hoeveel er al in de juiste positie staan. Voor alle duidelijkheid: als de computer de rij 1 5 4 7 in 'gedachten' heeft en u noemt 4 5 3 8 dan meldt de computer 'ER ZIJN TWEE GETALLEN UIT DE CODE BIJ' namelijk 4 en 5, en 'DAARVAN STAAN ER 1 OP DE GOEDE PLAATS (nl. 5). Mocht u het oorspronkelijke spel nog niet hebben dan zal de ervaring met onderstaand programma u ongetwijfeld naar de winkel doen rennen. Bij het 'in gedachten nemen' van vier cijfers kan de computer ook dubbele cijfers inlezen. De computer werkt met de getallen 0 t/m 7.

Programma

```
10 REM MASTERMIND
20 GOSUB 100:INITIALISEREN
30 GOSUB 500:REM INVOER
40 GOSUB 1000:REM ANTWOORD BEREKENEN
50 GOSUB 1200:REM UITVOER
60 IF GOED<>4 THEN GOTO 30
70 END
100 REM INITIALISATIE
110 DIM M(3,3)
120 FOR A=0 TO 3
130 M(0,A)=INT(RND(0)*8)
140 FOR B=1 TO 3:M(B,A)=0:NEXT B
150 NEXT A
160 RETURN
500 REM INVOER
510 PRINT "VOER VIER GETALLEN IN: G1,G2,G3,G4":INPUT
G1,G2,G3,G4
520 M(1,0)=G1:M(1,1)=G2:M(1,2)=G3:M(1,3)=G4
530 RETURN
1000 REM ANTWOORD
1010 GOED=0:FOUT=0
1015 FOR A=0 TO 3:M(2,A)=0:M(3,A)=0:NEXT A
1020 REM GOEDE PLAATS
1030 FOR A=0 TO 3
1040 IF M(0,A)=M(1,A) THEN GOED=GOED+1:M(2,A)=1:M(3,A)=1
1050 NEXT A
1060 REM FOUTE PLAATS
1070 FOR A=0 TO 3
1080 FOR B=0 TO 3
1090 IF M(1,A)=M(0,B) AND NOT (M(2,B) OR M(3,A)) THEN
FOUT=FOUT+1:M(2,B)=1:M(3,A)=1
1100 NEXT B
```

```
1110 NEXT A
1120 RETURN
1200 REM UITVOER
1210 IF GOED=4 THEN PRINT "GEFELICITEERD!":PRINT "U
HEEFT HET GERADEN":RETURN
1220 PRINT "UW GOK WAS: ";G1;G2;G3;G4
1230 PRINT "ER ZIJN ";GOED+FOUT;" GETALLEN UIT DE CODE
BIJ"
1240 PRINT "DAARVAN STAAN ER ";GOED;" OP DE GOEDE
PLAATS"
1250 RETURN
```

Voorbeeld:

```
VOER VIER GETALLEN IN: G1,G2,G3,G4
? 0,1,2,3 UW GOK WAS: 0123
ER ZIJN 2 GETALLEN UIT DE CODE BIJ
DAARVAN STAAN ER 2 OP DE GOEDE PLAATS
VOER VIER GETALLEN IN: G1,G2,G3,G4
?

     enzovoort
```

Zoeken

'Zoeken' is een zeer eenvoudig spelletje. U dient steeds een X- en Y-coördinaat op te geven om op deze wijze een onzichtbaar doel op te sporen. Als getallen mag u de getallen 1 t/m 20 invoeren. Het programma houdt precies bij hoeveel beurten nodig zijn om het doel te vinden. Na iedere opgave van een X- en Y-coördinaat meldt de computer zinvolle informatie, namelijk of we meer naar rechts of links en tevens of we meer naar boven of naar beneden moeten zoeken.

Programma

```
10 REM ZOEKEN
20 P=INT(RND(0)*20+1)
30 Q=INT(RND(0)*20+1)
40 PRINT "VOER X IN:";:INPUT X
50 PRINT "VOER Y IN:";:INPUT Y
60 Z=Z+1
100 IF P-X>0 THEN PRINT "NAAR RECHTS"
110 IF X-P>0 THEN PRINT "NAAR LINKS"
120 IF Q-Y>0 THEN PRINT "NAAR BOVEN"
130 IF Y-Q>0 THEN PRINT "NAAR BENEDEN"
140 A=(P-Y)*(P-X)+(Q-Y)*(Q-Y)
150 IF A>0.1 THEN 40
160 PRINT "GESLAAGD IN ";Z;" BEURTEN"
170 END
```

Voorbeeld:

```
VOER X IN:?5
VOER Y IN:?10
NAAR RECHTS
NAAR BOVEN
VOER X IN:?
enz.
```

Differentiëren

Het hier beschreven programma bepaalt de afgeleide van een functie in een aan te wijzen punt door rond dat punt twee functiewaarden te bepalen. De functie is als subroutine gegeven (regel 300 - regel 320).
Als delta-x is hier de waarde 0.001 genomen. Uiteraard hangt zo'n keuze af van de functie waarvan men de afgeleide op een bepaald punt wil benaderen.
Het programma is alleen bedoeld om voor een bepaalde functie snel afgeleiden te bepalen. Een nog betere weg is uiteraard het differentiëren van de functie zelf. Zo is de afgeleide van de gegeven functie $3x^2$ en voor x=5 vinden we zo de waarde 3*25=75.

Programma

```
10 REM DIFFERENTIEREN
100 GOSUB 500:REM INVOER
110 GOSUB 600:REM BEREKENING
120 REM UITVOER
130 PRINT "DE AFGELEIDE VOOR X=";
140 PRINT PUNT
150 PRINT "is: ";AFGELEIDE
200 END
300 REM FUNCTIE
310 F=X*X*X
320 RETURN
500 REM INVOER
510 PRINT "VOER DE X-WAARDE IN: "
520 INPUT PUNT
530 RETURN
600 REM BEREKENING
610 X=PUNT-1.0E-03
620 GOSUB 300
630 HULP=F
640 X=PUNT+1.0E-03
650 GOSUB 300
660 AFGELEIDE=(F-HULP)*500
670 RETURN
```

Voorbeeld:

```
VOER DE X-WAARDE IN:
?5
DE AFGELEIDE VOOR X=5
IS: 75.0005
```

De nauwkeurigheid van het antwoord hangt sterk samen met de gekozen delta-x. Probeer maar eens een aantal verschillende waarden.

Eeuwkalender

Dit programma bepaalt voor iedere datum van deze en de vorige eeuw de bijbehorende dag. Zo kunt u eindelijk eens nagaan op wat voor dag u geboren bent. Alle gegevens dienen als getal te worden ingevoerd. Zo heeft het getoonde voorbeeld betrekking op 4 juli 1984.

Programma

```
10 REM EEUWKALENDER
15 DIM EINDE$(1)
20 GOSUB 100:REM INVOER
30 GOSUB 200:REM BEREKENING
40 GOSUB 300:REM UITVOER
50 PRINT "EINDE (J/N)";:INPUT EINDE$
60 IF EINDE$<>"J" THEN 20
70 END
100 REM INVOER
110 PRINT "WELKE DAG";:INPUT DAG
120 PRINT "WELKE MAAND";:INPUT MAAND
130 PRINT "WELK JAAR";:INPUT JAAR
140 RETURN
200 REM BEREKENING
210 IF MAAND<=2 THEN JAAR=JAAR-1:MAAND=MAAND+12
220 E=INT((13/5)*(MAAND+1))
230 F=INT(5*JAAR/4):G=INT(JAAR/100)
240 H=INT(JAAR/400):T=DAG+E+F-G+H
250 DA=INT((T/7-INT(T/7))*7+0.1)
260 RETURN
300 REM UITVOER
305 IF MAAND>12 THEN MAAND=MAAND-12:JAAR=JAAR+1
310 PRINT DAG;"-";MAAND;"-";JAAR;" ";
320 PRINT "IS EEN ";
330 ON DA+1 GOTO 340,350,360,370,380,390,400
340 PRINT "ZATERDAG.":RETURN
350 PRINT "ZONDAG.":RETURN
360 PRINT "MAANDAG.":RETURN
370 PRINT "DINSDAG.":RETURN
380 PRINT "WOENSDAG.":RETURN
390 PRINT "DONDERDAG.":RETURN
400 PRINT "VRIJDAG.":RETURN
```

Voorbeeld:

```
WELKE DAG?4
WELKE MAAND?7
WELK JAAR?1946
4-7-1946 IS EEN WOENSDAG.
```

Lineaire hypotheek

Bij een lineaire hypotheek is het jaarlijks te betalen aflossingsdeel over de gehele looptijd constant. Het rentedeel neemt echter ieder jaar af, waardoor het jaarlijks te betalen bedrag aan rente en aflossing ook afneemt.

In formule: rentedeel n = (HB-A∗(n-1)∗i)

met: n = index voor n-de jaar
HB = hypotheekbedrag
A = aflossingsdeel
i = percentage

Bij een variant op de lineaire hypotheek neemt het aflossingsdeel toe, naarmate de hypotheek vordert (hypotheek met klimmende aflossing). Bijvoorbeeld:

1e periode van 5 jaar 2%
2e periode van 5 jaar 2,5%
3e periode van 5 jaar 3%
4e periode van 5 jaar 3,5%
5e periode van 5 jaar 4%
6e periode van 5 jaar 5%

Beide mogelijkheden zijn met dit programma te berekenen.

Programma

```
10 REM LINEAIRE HYPOTHEEK
20 PRINT "↰"
30 REM INVOERGEGEVENS
40 GOSUB 80
50 REM PRINT RESULTATEN
60 GOSUB 140
70 END
80 REM INVOER GEGEVENS
90 PRINT "HYPOTHEEK: ";:INPUT HYP
100 PRINT "LOOPTIJD: ";:INPUT JAAR
110 PRINT "RENTE/JAAR: ";:INPUT RENTE
120 PRINT "AFLOSSINGSDEEL: ";:INPUT AFLOSDEEL
130 RETURN
140 REM PRINT RESULTATEN
150 PRINT
160 PRINT " HYPOTHEEK : FL ";HYP
170 PRINT "  LOOPTIJD :    ";JAAR;" JAAR"
180 PRINT "RENTE/JAAR :    ";RENTE;" %"
190 PRINT " AFLOSDEEL : FL ";AFLOSDEEL
200 PRINT
210 PRINT "JAAR, RENTE, TOTAAL:"
220 PRINT
240 I=RENTE/100
250 FOR T=1 TO JAAR
```

```
260 RD=I*HYP
270 AN=RD+AFLOSDEEL
280 HYP=HYP-AFLOSDEEL
290 PRINT T;", ";
300 PRINT RD;", ";
310 PRINT AN
320 NEXT T
330 PRINT
340 PRINT "RESTHYPOTHEEK: ";HYP
350 PRINT
360 RETURN
```

Voorbeeld:

Iemand leent *f* 150.000,- op basis van een lineaire hypotheek. Over de eerste 5 jaar bedraagt het aflossingsdeel 2% dus *f* 3000,- per jaar.
Het rentepercentage is 8%. Het aflossingsschema voor de eerste periode is dan:

```
HYPOTHEEK: ?150000
LOOPTIJD: ?5
RENTE/JAAR: ?8
AFLOSSINGSDEEL: ?3000

     HYPOTHEEK: FL150000
      LOOPTIJD: 5 JAAR
    RENTE/JAAR: 8 %
     AFLOSDEEL: FL3000

     JAAR RENTE       TOTAAL

     1,    12000,     15000
     2,    11760,     14760
     3,    11520,     14520
     4,    11280,     14280
     5,    11040,     14040

     RESTHYPOTHEEK: 135000
```

Priemgetallen

Priemgetallen zijn gehele getallen die bij deling alleen een geheel getal opleveren als we door het getal zelf of door 1 delen. Zo is 5 bijvoorbeeld een priemgetal, dit getal is immers alleen deelbaar door 1 en 5. Priemgetallen zijn voor wiskundigen van formaat steeds een geweldige uitdaging geweest. Met name de vraag of men een zeker verband tussen de opeenvolgende getallen zou kunnen ontdekken heeft steeds velen geïntrigeerd. In het gezaghebbende blad Scientific American treft men hierover met de regelmaat van de klok (of met de regelmaat van 'priemgetallen'...) artikelen aan.
Het volgende programma berekent alle priemgetallen tussen twee opgegeven grenzen. Het aantal waarden dat men kan verkrijgen zal uiteindelijk door de beschikbare geheugenruimte worden bepaald.

Programma

```
10 REM PRIEMGETALLEN
20 REM INPUT
30 PRINT "*** PRIEMGETALLEN ***"
40 PRINT "BEGINWAARDE: ";:INPUT BEGIN
50 PRINT "EINDWAARDE: ";:INPUT EIND
60 DIM PR(EIND-BEGIN)
65 FOR Q=0 TO EIND-BEGIN:PR(Q)=0:NEXT Q
70 PRINT "TUSSENLIGGENDE PRIEMGETALLEN:"
80 REM BEREKENING
90 GOSUB 150
100 FOR A=0 TO EIND-BEGIN
110 IF PR(A)=0 THEN PRINT (A+BEGIN);:PRINT "  ";
120 NEXT A
130 PRINT
140 END
150 REM BEREKENING
160 X1=BEGIN
170 P=P+1+( NOT (P=2))
180 IF P>SQR(EIND) THEN GOTO 260
190 IF X1/P-INT(X1/P)=0 THEN GOTO 220
200 X1=X1+1:GOTO 190
210 REM WEGSTREPEN PRIEMGETALLEN
220 IF X1>EIND THEN GOTO 160
230 IF X1>P THEN PR(X1-BEGIN)=1
240 X1=X1+P
250 GOTO 220
260 RETURN
```

Voorbeeld:

```
*** PRTEMGETALLEN ***

BEGINWAARDE: ?10
EINDWAARDE: ?50
TUSSENLIGGENDE PRIEMGETALLEN:
11  13  17  19  23  29  31  37  41  43  47
```

Dieren raden

Dit spel toont de mogelijkheden van de Atari als het gaat om manipulaties met letters. Iedere keer als u het commando RUN geeft, verschijnt een naam van een dier op het beeldscherm... alleen de letters staan niet in de vertrouwde volgorde. Alle namen van de dieren zijn in feite in de string-variabele P$ verborgen. Bij het intoetsen dient u alle letters direct na elkaar in te toetsen, dat wil zeggen er mogen geen spaties worden ingevoegd.

Programma

```
10 REM DIEREN RADEN
100 REM INITIALISATIE
110 DIM P$(50),B$(5),G$(5),X$(5)
120 P$="LOHCSARBOCREDDAGEILVREGYTWUEELLEREMKEONSSRA
ABFARIG"
200 REM BEREKENING
210 C=INT(RND(0)*9)*5+1:X$=P$(C,C+4)
220 FOR K=5 TO 1 STEP -1
230 R=INT(RND(0)*K+1)
240 G$(K,K)=X$(R,R)
250 IF R=K THEN 270
260 X$(R,K-1)=X$(R+1,K)
270 NEXT K
300 REM INVOER
310 PRINT G$;" IS EEN:";:INPUT B$
400 REM CONTROLE
410 FOR K=1 TO 5
420 X$(K,K)=B$(6-K,6-K)
430 NEXT K
440 IF X$=P$(C,C+4) THEN PRINT "GOED":GOTO 460
450 PRINT "FOUT":GOTO 300
460 END
```

Voorbeeld:

```
GELVI IS EEN: ?LEVIG
FOUT
GELVI IS EEN: ?VLIEG
GOED
```

Integreren

Met het onderstaande programma kan men de integraal van een functie F(X) tussen de grenzen A en B berekenen.
Allereerst wordt het aantal stappen opgevraagd waarmee de integraal zal worden berekend. De functie is in een subroutine ondergebracht: regel 300 t/m 320. In dit geval is als voorbeeld de functie X^3 dat wil zeggen X*X*X opgenomen.

Programma

```
10 REM INTEGREREN
100 GOSUB 500:REM INVOER
110 GOSUB 600:REM BEREKENING
120 REM UITVOER
130 PRINT "DE INTEGRAAL TUSSEN DE GRENZEN"
140 PRINT "A=";A
150 PRINT "B=";B
160 PRINT "IS: ";SOM
200 END
300 REM FUNCTIE
310 F=X*X*X
320 RETURN
500 REM INVOER
510 PRINT "AANTAL STAPPEN"
520 INPUT STAPPEN
530 PRINT "A=";
540 INPUT A
550 PRINT "B=";
560 INPUT B
570 RETURN
600 REM BEREKENING
610 STAP=(B-A)/STAPPEN
620 SOM=0
630 FOR X=A+0.5*STAP TO B-0.5*STAP STEP STAP
640 GOSUB 300
650 SOM=SOM+STAP*F
660 NEXT X
670 RETURN
```

Voorbeeld:

```
AANTAL STAPPEN
?20
A=?0
B=?2
DE INTEGRAAL TUSSEN DE GRENZEN
A=0
B=2
IS: 3.995
```

N.B. In dit voorbeeld werd de integraal van de functie $F(X)=X^3$ uitgerekend, en wel tussen de grenzen 0 en 2. Wil men bijvoorbeeld de integraal van de functie $2X^2+3X$ uitrekenen (tussen de grenzen A en B) dan dient men alleen regel 310 te vervangen, namelijk door:

```
310 F=2*X*X+3*X
```

Trefwoordenregister

A

aanhalingstekens 26, 46
ABS 24, 150
ADR 151
adresseermethoden 136
afkorten 35
afronden 43
alternatieve karakterset 115
AND 61
animated graphics 109
argument 24
array 73, 79
ASC 95, 152
ASCII-codes 95, 145
ASSEMBLER 19
ATN 24, 151

B

BACKSPACE-toets 39
BASIC 13, 17
BASIC-commando's 152
BASIC-functies 150
BASIC-instructies 152
BASIC-interpreter 13
binaire getallen 141
bit 12
BREAK 41
bubble sort-methode 84
BYE 35 , 153
byte 12

C

CAPS 40
cartridge poort 15
cassetterecorder 13
centrale verwerkingseenheid 10
CHR$ 45, 152
CLOAD 154
CLOG 150
CLOSE 154
COLOR 119
commando 33
CONT 154
conversies 116
CONTROL-toets 40
COS 24, 151
CPU 11
CSAVE 154
cursor 21
cursor-besturingstoetsen 40
CVE 11

D

DATA 154
DATA-lijst 52
DEG 151
DELETE 40
deling 22
DIM-instructie 46, 79
dollarteken 45
DOS 155
DRAWTO-instructie 123
dubbele punt 35

E

eindwaarde 76
elementen 85

END-instructie 30, 155
ESC-toets 41
EXP 24, 151
exponent 44

F

faculteit 90
floppy disk unit 13
FOR...NEXT-instructie 72, 73, 155
foutboodschappen 37, 165
FRE 152
functies 24
functiewaarden 24

G

gehele getallen 43
geheugen 12, 162
geheugenadressen 114, 133, 163
gekoppelde lijsten 83
geluid 102
geluidsgeneratoren 102
geluidssterkte 102
geneste lus 77
geneste subroutines 91
GET 155
getallen 142
getalvariabelen 45
grafische modes 109
graphics 109
GRAPHICS 110
grenswaarde 76
GOSUB-instructie 88, 155, 156
GOTO 58, 155, 157
grafische mogelijkheden 119
grafisch venster 114

H

hardware 10, 17
HELP 36
hexadecimale getallen 131
hoofdnaam 80

I

IF...THEN-instructie 58, 155
index 80
INPUT-instructie 48, 156
instructies 30
integer-getallen 43
INT-functie 24, 43, 151

K

karakterset 115
kleurcodes 110
kleuren 110, 160
kleurregisters 110, 111
komma 22, 54

L

labels 66
LEN-functie 94, 152
LET 28
lichtsterkte 110
LIST 33, 34, 156
LOAD 156
LOCATE-instructie 117, 127
LOG 24, 151
LOGO 19
LPRINT 156
lus 76

M

machine-codes 130
machine-instructies 130, 170
mantisse 44
maximum 83
microprocessor 12
minimum 83
MODE 109, 128, 160, 162
monitor 15

N

negatieve getallen 142
NEW 33, 34, 156
NEXT 156
niet-gehele getallen 43
NOT 62
NOTE 156
nul-string 47

O

ON 156, 157
ON...GOSUB-instructie 92
ON...GOTO-instructie 71
OPEN 157
OPTION 113
OR 61

P

PASCAL 19
PEEK-instructie 113, 152
PILOT 19
pixels 109
PLOT-instructie 120
POINT 151
POKE-instructie 112
POP 157
POSITION-instructie 117
printer 11, 14
PRINT-instructie 21, 54, 157
6502-processor 135
programma 11, 27
punt 22
puntkomma 54
PUT 158

R

RAD 151
RAM-geheugen 12
READ 154, 158
READ/DATA-instructie 52
reële getallen 43
registers 12
regelnummer 29
relatie-symbolen 99
relatietekens 61
REM-instructie 55, 158
RESET 36
RESTORE-instructie 53, 154, 158
RETURN 21, 88, 156
RND 151
ROM-geheugen 12
ROM-modules 19
RUN 33, 158

S

SAVE 158
scheidingsteken 35, 54
schermfametingen 162
schrifttekens 145
schuine deelstreep 23
SELECT 36, 113
SETCOLOR 110
SGN 24, 151
SHIFT-toets 33
SIN 24, 151
slicing 96
software 10, 16
sorteren 84
SOUND-instructie 102
speciale functies 151
SQR 24, 151
standaard 115
START 36, 113
startadres 136
STEP 75
STOP-instructie 89, 158
STR$ 95, 152
string 45, 94
string-functies 94, 152
string-variabele 45
stroomdiagram 68
subroutine 88
substring 97
symbolen 145

T

TAB 41
teken ; 54
teken ^ 25
teken / 23
teken * 22
tekstmodes 109, 114
tekstvenster 114
TO 158
tonen 102
toonhoogte 102
TRAP 158
tussenvoegen van een regel 38
twee-dimensionale array 85

U

uitbreidingspoort 15

V

VAL 96, 152
variabele 28
variabelenaam 28
vermenigvuldigen 22
vervorming 106
vervormingsfactor 102
verwijderen van een regel 39
vierkantswortel 24
vraagteken 49

W

wiskundige functies 150
wijzigen van een regel 39

X

XIO-instructie 125

Z

zoekacties 83

Behalve de welbekende spelcomputers heeft Atari inmiddels een aantal uitstekende huiscomputers in het programma opgenomen. De 600XL en de 800XL zijn hiervan de meest bekende voorbeelden. Van Albert Sickler mag intussen bekend worden verondersteld dat hij een kenner is op het gebied van microcomputers en BASIC, maar vooral in staat is om deze zaken voor iedereen begrijpelijk te maken.
De inhoud van het boek is als volgt samen te vatten:
In de eerste hoofdstukken wordt gestart met algemene informatie over de machines gevolgd door instructies betreffende het aansluiten, het aanzetten en het verkennen van het toetsenbord.
Hierna volgt een uitgebreide cursus BASIC, speciaal gericht op Atari-BASIC. Daarna volgen de appendices die in deze serie een zeer belangrijk aandeel hebben.
De enigszins gevorderde computergebruiker kan direct naar deze appendices grijpen zonder zich eerst door de inleiding en de cursus te moeten heenworstelen. Men vindt hierin een overzicht van zowel de hardware- als de de software-mogelijkheden van beide machines.
Ten slotte vindt men achterin het boek een verzameling BASIC-programma's die de mogelijkheden van de Atari-computers het best illustreren. De gebruiker kan deze programma's direct intoetsen en ermee werken.
Er wordt voor dit boek van de lezer geen bijzondere voorkennis van computers en informatica verwacht.

ISBN 90 201 1763 7

ALBERT SICKLER

ATARI 600XL/800XL